Almost Free PC and Mac Software

Computing Now!

Canada's Personal Computing Magazine

$3.50
MM70247

July 1986

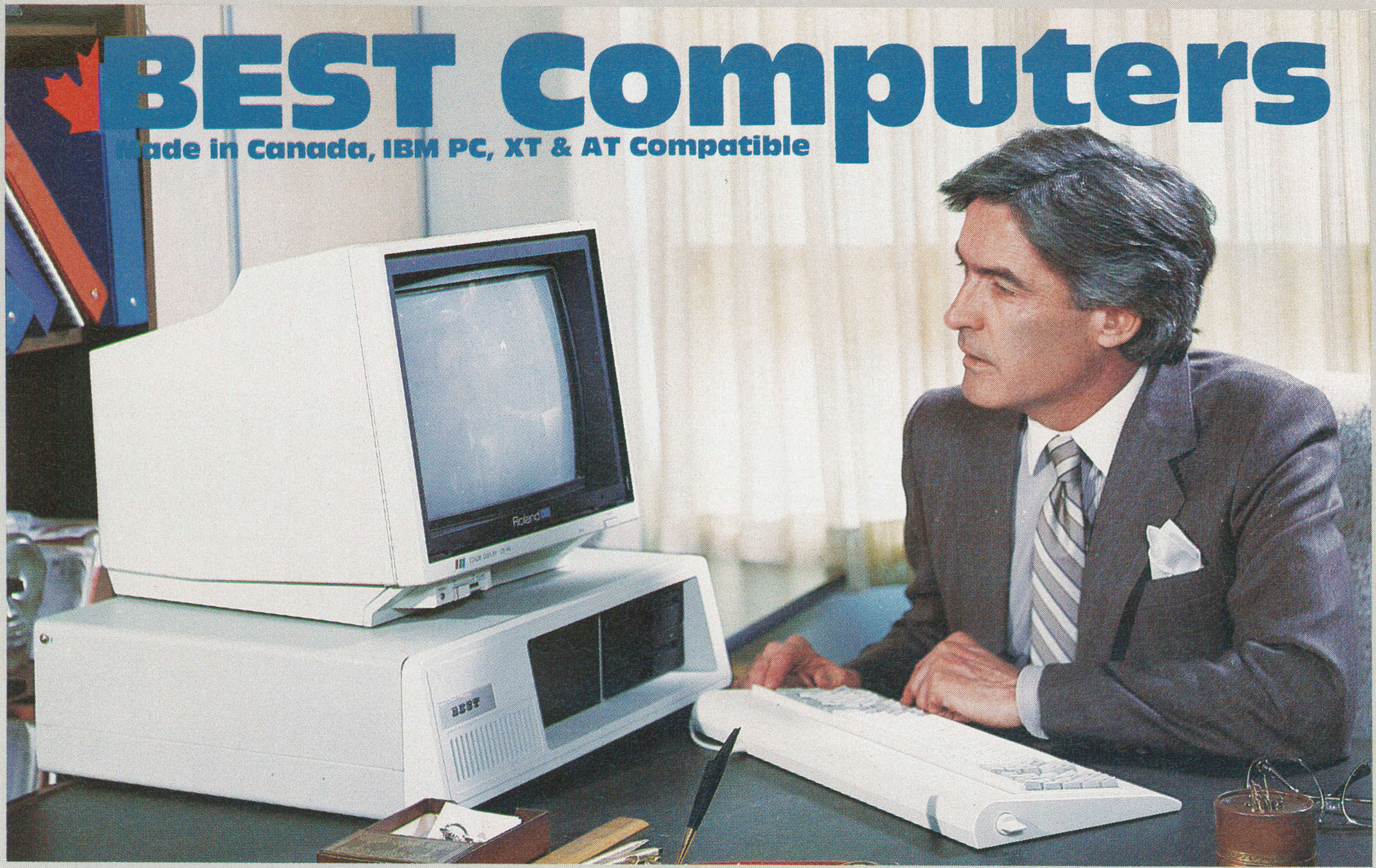
Creativity
The Ultimate Tool

- Kawai K3 MIDI Keyboard
- Digging Into the Mac
- Atari ST Software
- More

Also

WordStar Patches

0 57438 70247 07



Monitor not included. IBM, IBM PC, IBM XT and IBM AT are registered trademarks of IBM Canada Ltd.

Features common to BEST MK II, MK III & MK IV

- Superb IBM PC & XT Compatibility
- Canadian Made
- 256K RAM Standard minimum (uses 41256K RAM chips)
- Expandable on board to 640K RAM
- Parallel Port (for printer)
- Serial Port (for communications)
- Real Time Clock/Calendar (with Battery Backup) Presocketed for optional co-processor — such as 8087 Math Processor
- IBM Compatible Keyboard
- Reset Switch
- Phoenix BIOS as used in many major brand IBM compatible systems.
- 150 Watt Power Supply which will handle a Hard Drive without upgrade.
- 7 Slots
- Flip top case
- 2 Slimline DS,DD 5¼", 360K Disk Drives
- Colour Video (RGB & Composite) Board or Hi-Res Monochrome Card (customer choice)
- Disk Controller Board

Warranty

We have such confidence in the time tested reliability of the BEST that we offer a ten (10) month warranty which is way above the industry standard. On-site service plan available at extra cost Nation-wide through 3M of Canada Ltd.

BEST MK II. Regular $1695. This Month:

Two 350K DS,DD disk drives, Serial and Parallel Ports, Real Time Clock, Phoenix BIOS, uses 8088 processor — full specifications given above.

$1595.00

TEN MONTHS WARRANTY

Other Configurations:

With 640K RAM	$1695
With 10 Meg Hard Drive/1 Floppy/256K	$2495
With 10 Meg Hard Drive/1 Floppy/640K	$2595
With 20 Meg Hard Drive/1 Floppy/256K	$2795
With 20 Meg Hard Drive/1 Floppy/640K	$2895

4.77/8MHz DUAL SPEED

The FAST BEST MK III

As BEST MK II plus speed selectable: 4.77 and 8MHz (most software will run on the higher speed), uses 8088-2 processor.

$1895.00

TEN MONTHS WARRANTY

Other Configurations:

With 640K RAM	$1995
With 10 Meg Hard Drive/1 Floppy/256K	$2795
With 10 Meg Hand Drive/1 Floppy/640K	$2895
With 20 Meg Hard Drive/1 Floppy/256K	$2895
With 20 Meg Hard Drive/1 Floppy/640K	$2995

SPECIAL

SUPER-FAST BEST MK IV

As BEST MK III plus TRUE 16-Bit machine, 8086-2 processor, IBM compatible 8-Bit I/O channel bus, even faster than MK III due to 16-Bit architecture. With 256K standard.

REGULAR $2400

$2095.00

TEN MONTHS WARRANTY

Other Configurations:

With 640K RAM	$2195
With 20 Meg Hard Drive/1 Floppy/640K	$3195

The BEST Mk IV Main board processes data approx. 2.1 times faster than the IBM PC/XT or BEST Mk II.

IBM AT Compatible

AVT-286

Superb IBM AT compatibility, 640K RAM, Two 5.25 in disk drives (one high density 1.2 Megabyte, one standard 360K), serial and parallel ports high quality keyboard, keyboard lock and status monitor.

$3495.00

TEN MONTHS WARRANTY

With 20 Megabyte fast stepping Hard Disk and Controller:

$4950.00

40 Megabyte systems also available at reasonable prices.

Exceltronix

319 College Street, Toronto, Ont. M5T 1S2
(416) 921-8941. Order line only 1-800-268-3798
Ottawa, 217 Bank Street (613) 230-9000
Visa, Mastercard and American Express accepted.

Circle No. 1 on Reader Service Card

Not the Cheapest — Just the BEST!

The flood of Far Eastern IBM Clones has brought about significant price reductions in the market. These new low price levels may make us seem uncompetitive but before you make a judgement based solely on price, **we ask you to make a fair comparison**; we believe that this will still show the **BEST is the BEST value.**

- **We have been offering a 300 day (ten month) warranty** for a long while now yet we know of no clone that has moved to equal or better this. Why? We can do this because we manufacture and test to a **very high standard** and **stand behind our products.**
- Take just one example of quality — we could shave the price noticeably by using cheap internal connectors, we can buy at a fraction of the price of the products we use because we know of the problems that can occur.
- Our floppy drives are either **Panasonic or Toshiba**, truely well made, quality drives that last — again, we could reduce our price by using cheaper drives that are engineered to lower standards.
- We use **Seagate hard drives: they stand behind their products so that we can.** We could reduce the price of a hard drive system literally hundreds of dollars by using the cheapest available but your information is important and reliability here is something we do not want to compromise on.
- All BEST systems undergo a **40 minute manual test** followed by a **48 hours automatic continuous test** where the hard and floppy drives and memory are tested for hundreds of cycles. When this is completed **a further 40 minutes of manual tests** are made — all so that the customer is unlikely to experience the problems which would not surface without this expensive testing.
- We use the **latest chip technology**. For example 41256 memory chips — several systems cannot handle anything bigger than a 4164.
- This attention to detail and quality runs through the entire engineering and production of the BEST range. This quality does **not show** on the outside — it shows in the repeat and recommended business we receive. We have been around a lot longer than most of our competitors and will be around when many of them have gone. **We have to stand by our products — and we do.**

Eugen F. Hutka

Eugen Hutka
President

BEST MK II
Package Deal

- **BEST MK II 256K (as on opposite page at $1595)**
- **Copal Printer (SC1200L) 120 c.p.s., 80 columns**
- **ZVM1220 or ZVM1230 Monitor**
- **Printer and Video cables**

Limited Time Offer. Special price applies to this package only.

$1895.00

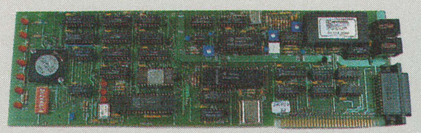
A $200 saving

BEST MODEM

The BEST modem is a smart 1200/300 direct connect modem. It can either be a stand-alone unit in which case it requires a small wall adaptor, or it plugs in one of the IBM slots. When used as a stand-alone unit, the modem looks like a Hayes 1200 Smart Modem, that is, it emulates the same instruction set. When it is used in the IBM, it looks like an intelligent serial communications port which also supports a super-set of the Hayes instruction set.

The modem supports auto-dial, auto-answer, and auto-speed select directly from software control. The modem also has a speaker so that aural monitoring of the call is possible. There are also LED monitors so that the state of the modem can always be known. These LEDs are: Modem Ready, Auto-Answer enabled, Carrier Detected, Transmitting, Receiving, Data Set ready.

Software packages such as Crosstalk, PC-talk, and Hayes' Smartcom II also will run with this modem.

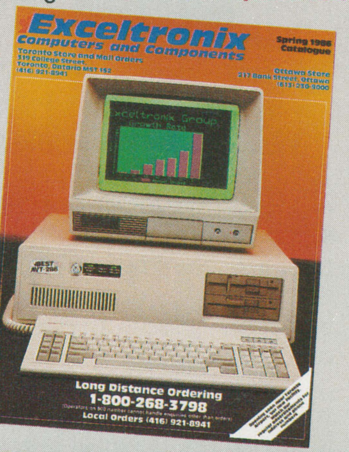


Board not exactly as illustrated.

300 Baud **$179.00**
300/1200 Baud . **$249.00**
300/1200 Baud, Stand-Alone Unit, Attractively Packaged **$299.00**

Call or Write for our Full Colour Catalogue

Our Spring 1986 catalogue gives full specifications on our IBM AT and XT compatibles, AVT-286, MK II, Mk III and MK IV computers plus detailed specifications, with prices on our other products, made or distributed by us.



Colour Monitor

TVM Model MD-3A
- 14 inch
- Multi-Display Amber/Green/ Full Colour
- Compatible with PC/XT/AT using RGBI TTL level
- 640x400 interlace
- One Year Warranty

Only $599.00

Copal Printers

- Epson Compatible • Supports both IBM and Apple systems
- Centronics Parallel Interface • Both Draft and Near Letter Quality (NLQ) modes

SC1200
- 120 cps draft quality
- 24 cps NLQ
- NLQ available in all print sizes
- 11 inch carriage
- Pin and friction feeds
- One Year Warranty

$329.00

SC1500 $499.00
- 180 cps draft quality
- 36 cps NLQ
- 3k Buffer

SC5500 $699.00
- 15 inch carriage
- 3K Buffer

Exceltronix

319 College Street, Toronto, Ont. M5T 1S2
(416) 921-8941. Order line only 1-800-268-3798
Ottawa, 217 Bank Street (613) 230-9000
Visa, Mastercard and American Express accepted.

Circle No. 2 on Reader Service Card

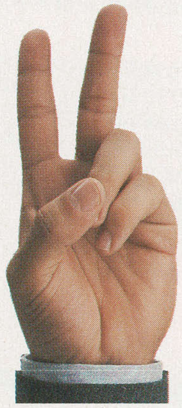
With one Hayes Transet 1000® you can do three things at once.

Think

You can keep working with your computer.

Communicate

While receiving electronic messages from your modem.



Print

While your printer is printing another job.

*Manufacturer's estimated retail price.

We call it triple tasking©. Transet 1000 helps your productivity, by allowing you to perform three separate computer tasks. Simultaneously. No more waiting for one task to be completed before you can begin another. And no more wasted time!

Transet 1000 keeps working for you—even when your computer is turned off. Because it has its own independent memory, Transet 1000 can serve as an electronic mailbox. Your files and incoming messages received through your modem remain stored overnight, or whenever you're away from the office. When you return, you can print out your mail without turning your PC on. Or, read it on your PC screen and print out only selected messages. You can even read your mail from any remote PC or terminal with a modem.

Before now, you would have had to buy several add-on devices to do all this. And *that* could be costly.

But no more. Transet 1000 does it all—for a surprisingly low price. It costs only $399* for the 128K model, which stores up to 90 pages. And only $549* for the 512K version, with up to 360 pages of storage.

So wait no more. If you need this flexibility in your operation, you should have a Transet 1000. See your authorized Hayes dealer for a demonstration. Or contact Hayes for information at (416) 671-0906. Hayes Microcomputer Products, Inc., 5955 Airport Road, Suite 200, Mississaug , Ontario, Canada L4V 1KQ.

Hayes®

Say yes to the future

Circle No. 3 on Reader Service Card

Computing Now!

Canada's Personal Computing Magazine

Volume 4 No. 4
July 1986

Published by
Moorshead Publications
1300 Don Mills Road,
Don Mills,
Toronto, Ont. M3B 3M8
(416) 445-5600

EDITOR
Steve Rimmer

ASSISTANT EDITOR
Marie Hubbs

DIRECTOR OF PRODUCTION
Erik Blomkwist

PRODUCTION
Dolph Loeb
Douglas Goddard
Naznin Sunderji

CIRCULATION MANAGER
Lisa Salvatori

ADVERTISING MANAGER
Denis Kelly
Toronto (416) 445-5600

Publisher: H.W. Moorshead; **Executive Vice-President:** V.K. Marskell; **Vice-President - Sales** A. Wheeler; **General Manager:** S. Harrison; **Controller:** B. Shankman; **Accounts:** P. Dunphy; **Reader Services:** N. Jones, L. Robson, M. Greenan, R. Cree, J. Fairbairn; **Advertising Copy:** H. Brooks; **Advertising Telemarketing:** W. Fleet

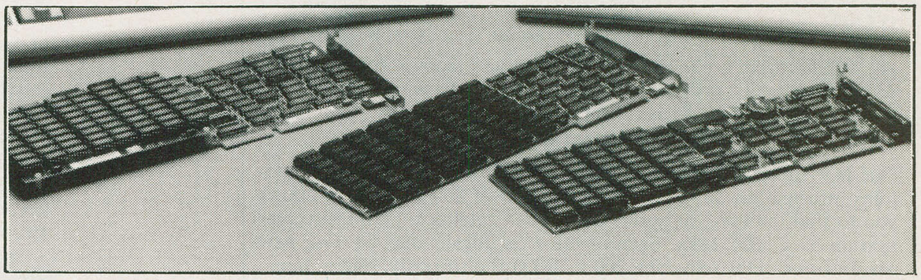
NEWSSTAND DISTRIBUTION
Master Media, Oakville, Ont.
PRINTED BY:
Heritage Press Ltd., Mississauga
SUBSCRIPTIONS
$22.95 (12 issues) $37.95 (24 issues)
Published 12 times a year
Outside Canada (US Dollars)
U.S. add $3.00 per year.
Other countries add $5.00 per year.
Moorshead Publications also publish Electronics Today, Computers in Education, and Pets Magazine

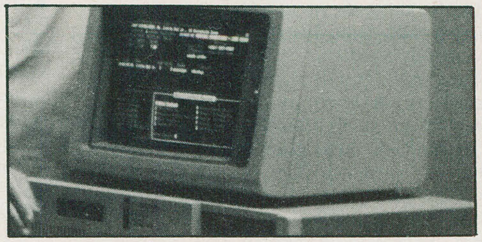
POSTAL INFORMATION

Second Class Mail Registration No. 5946. Mailing address for subscription orders, undeliverable copies and change of address notice is: Computing Now!, 1300 Don Mills Road, Don Mills, (Toronto), Ont. M3B 3M8

Printed in Canada ISSN 0823-6437.

Circulation independently audited by MURPHY and MURPHY, Chartered Accountants.





Stray Bits

COMPUTER PRESS

by Marie Hubbs

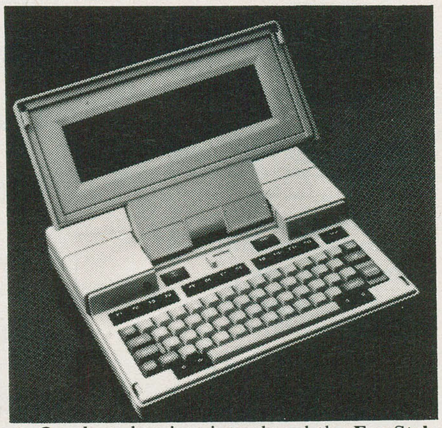
Portable Carry Overs

As always happens when we compile our various surveys, there are stragglers. Here's yet more **information on portables**, whether you wanted it or not!

● Weighing in at a mere four kilograms, the *Time* **Kookaburra** includes ninety-six kilobytes of user memory, and such ROM cartridge software as MS-DOS, communications, word processing, spreadsheet and personal planning, with external floppy drives optional.

Costing $2,521.00, the Kookaburra is available from Man-Tech Associates, 143 Dennis Street, Rockwood, Ontario N0B 2K0, telephone (519) 856-4565, or in the Toronto area (416) 456-2628.

Circle No. 24 on Reader Service Card



● *Quadram* has just introduced the **FreeStyle portable system**, consisting of the WriteStyle portable letter quality printer, and the KeyStyle 80, an "intelligent keyboard" which can function alone as a word processor or as a keyboard for either the Datavue portable computer or an IBM PC. Combined, the package weighs only nine pounds.

For pricing of the separate units or the whole system, contact Chevco Computing, 6581 Kitimat Road, Unit 14, Mississauga, Ontario L5N 3T5, or call (416) 821-7600.

● Just announced by *BackPac International*, Portable Solutions' PSDOS 3.00 software is an MS-DOS device driver package allowing expansion of storage capacities in 115 megabyte increments by adding additional optical drives as needed. These **modular drives**, which may be attached to most standard PC, XT, AT or compatibles, are five and one quarter inch write-once optical disk drives, using removable optical cartridges.

For further information, as well as Canadian pricing and availability, get in touch with J. Donald Speer, Vice President Sales/Marketing, 1701 Directors Boulevard, Suite 250, Austin, Texas 78744, telephone (512) 448-4965.

Circle No. 25 on Reader Service Card

● Transport ten or twenty megabytes of data from system to system in complete safety with the PortaFile **portable Winchester expansion system.** Useful for compatible portables as well as the IBM PC or XT, the PortaFile features interface connections for daisy chaining, a

built-in power supply, and XTREE file management software.

Pricing depends on configuration. For local availability, you can call the source: *Western Digital*, Enhanced Peripherals Division, 2300 Main Street, Irvine, California 92714, telephone (714) 863-0533.

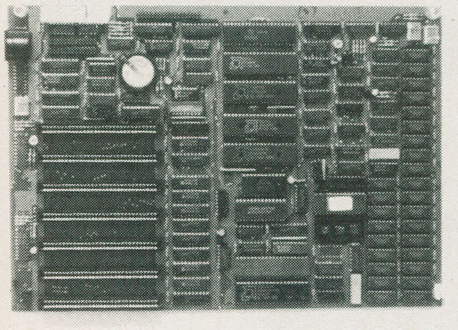
Circle No. 26 on Reader Service Card

On Board!

• A multi-function IBM PC XT/AT compatible **motherboard**, the Full House, is now available from *CPU Electronic Systems*. Retailing for $750.00, the Full House board includes a disk controller, 640K of RAM, calendar, battery backed-up clock, support for the 8087 co-processor, one serial and one parallel port. Doubling the speed of the IBM PC and compatibles, the Full House operates at eight megahertz.

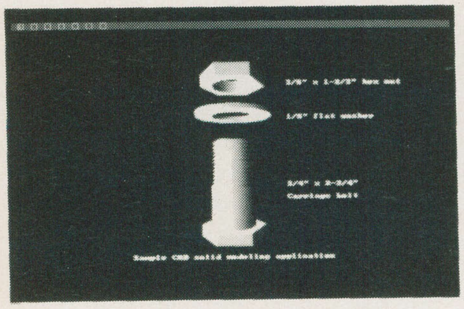
Manufactured in Canada, and available through CPU dealers, the Full House comes from CPU Electronic Systems Manufacturing, 2652 Slough Street, Mississauga, Ontario L4T 3T2, telephone (416) 673-7355.

Circle No. 27 on Reader Service Card



• The *SubLOGIC* **X-1 Graphics Board** brings high speed colour graphics and real-time animation to the IBM PC for just under three thousand dollars American, and includes custom language interfaces to Microsoft BASIC, C, Fortran and Assembler.

See you local dealer to arrange a demonstration, or contact SubLOGIC Corporation direct, at 713 Edgebrook Drive, Champaign, Illinois 61820, (217) 359-8482.



• *Pure Data* announced **two new multifunction boards** for the IBM PC/AT, both having a 16450 UART with 8250 UART emulation capabilities, and both being expanable to one megabyte of memory. The PDI1152SP has one serial and one parallel port and costs $869.00, while the PDI1152SS costs one hundred dollars more and has two serial ports.

For the nearest Canadian dealer, contact Pure Data at 860 Denison Street, Markham, Ontario L3R 4H1, telephone (416) 494-9563 or 475-3370.

Circle No. 28 on Reader Service Card

Musically Speaking

Two new MIDI boxes have been announced recently by *Roland*.

The MC-500 Micro Composer is a **real-time sequencer** which can be used to control any MIDI sound producing module, has four recording tracks and allows complex data editing with a two line LCD display. With internal memory storage for about thirty thousand notes, the MC-500 uses three and a half inch disks for external storage, and retails for just under two thousand dollars.

The lastest in the TR series of **rhythm composers**, the TR-505 can function as either a programmable drum machine or a MIDI sound source, with forty-eight preset rhythm patterns, and an equal number of programmable ones. Featuring tape interface, LCD and AC/DC operation, the TR-505 costs $525.00.

For the Roland dealer nearest you, contact Roland Canada Music, 1515 Matheson Boulevard, Units B11 and B12, Mississauga, Ontario L4W 2P5, (416) 625-4880, (514) 335-2009 in the Montreal area, or (604) 273-4453 in the Vancouver area.

Circle No. 29 on Reader Service Card

Neat STuff

• *Prospero Software* has recently added the **Pro Pascal language compiler** to its list of products for the Atari ST. A full and complete ANSI 770X3.97 standard Pascal compiler, Pro Pascal is available from most Atari dealers and distributors, and... like the Pro Fortran-77... costs about $150 U.S. and is not copy protected.

Further information can be obtained from Prospero Software, 190 Castelnau, London, SW13 9DH, England, telephone 011-441-741-8531.

• *MichTron*, mentioned in the ST article elsewhere in this issue, have introduced even more goodies.

Cornerman provides a host of **desktop tools** including a real-time clock, notepad, ASCII table, phone book, sixteen digit calculator and DOS window, all for $49.95 U.S.

Alt, with a cost of $29.95, is a macro utility for use with the Alt key.

The Animator handles up to 256 moving objects and 1024 **animated frames**, using objects and backgrounds painted by Neo or Degas, and it only costs $39.95 U.S.

Personal Money Manager, costing $49.95 U.S., **keeps accounts**, prints cheques and allows you to form projected budgets and reports of your own finances using the GEM system.

MichTron can be reached at 576 South Telegraph, Pontiac, Michigan 48053, telephone (313) 334-5700.

Next Month In Computing Now!

Canada's Personal Computing Magazine

Creating a PC

There are so many cases and keyboards and power supplies and disk drives and video boards and cables and connectors and controllers and mother boards and BIOS's and other bits floating around that we really *had* to find out whether they could all be put together to build a PC. In fact, not only can you get a really tight computer together doing this, but it can come out a bit cheaper. We'll be looking at the macinations of rolling your own system next month, and we'll check out a really splendid home grown mother board.

UltraVideo

In the next edition of Computing Now! we're going to look at the universe beyond the PC's colour card. There are some unspeakably slick video cards out there... we're going to check out several of them to see what you can really do with twelve zillion colours dancing across an infinite number of pixels.

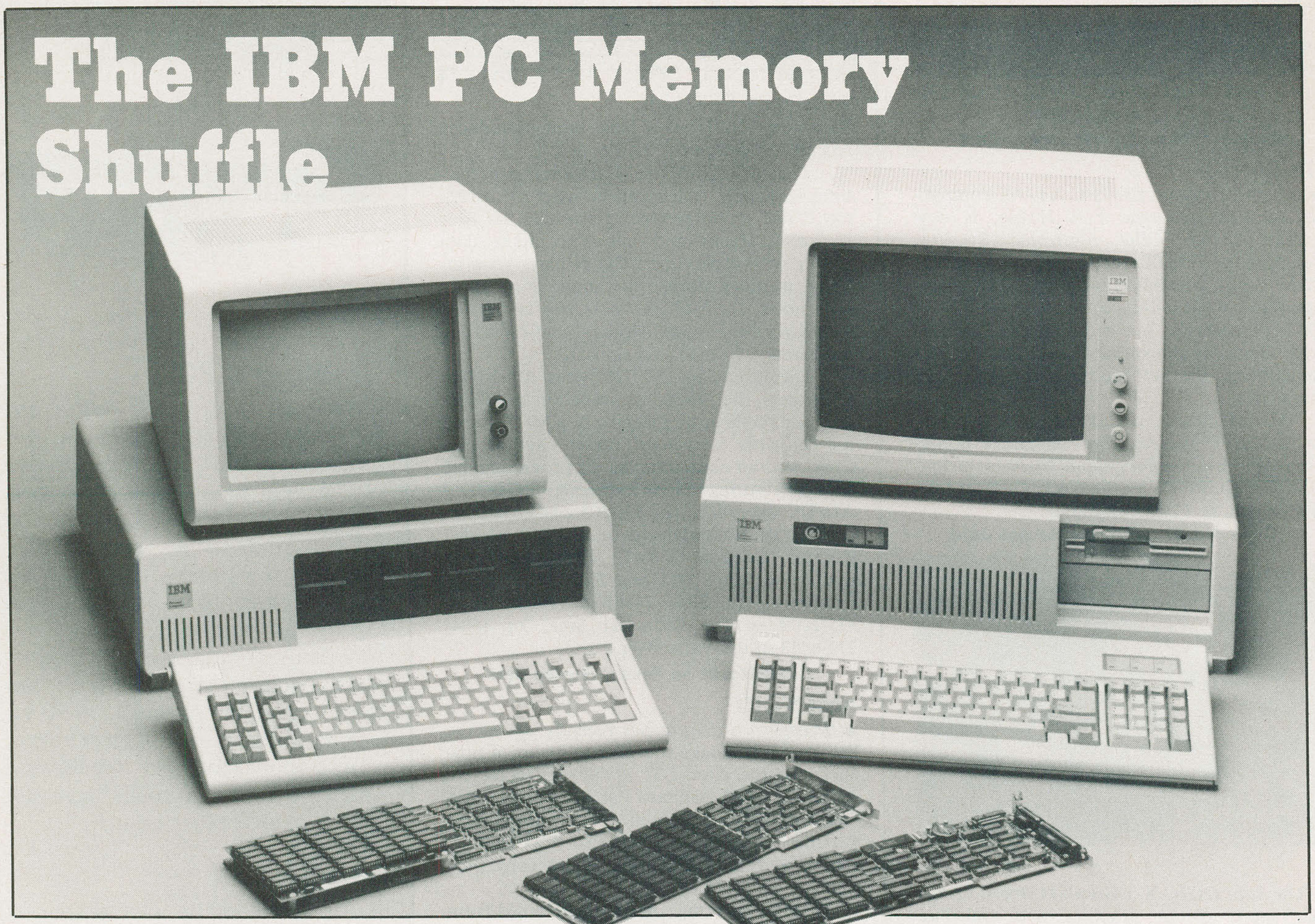
More Fractals

A few months ago we ran a couple of articles about MandelZoom fractal generators that just seemed to catch everyone's interest. In the next edition of CN! we'll be looking at some more sophisticated fractal programs, including one for the PC that illustrates the use of the 8087 from machine language.

These features are in an advanced state of preparation. However, in endeavouring to keep Computing Now! as up to the minute as possible we reserve the right to change the contents of this issue prior to going to press.

For Advertising and Subscription Information Call (416) 445-5600

The IBM PC Memory Shuffle



The extended memory capacity of PCs brings with it a fair number of complications. We'll check out some of the unscramblings herein.

by Steve Rimmer

One of the things that the sixteen bit PC systems have over the older eight bit computers that they largely ground into the dust of ages is their memory capacity. Those early computers could only access sixty-four kilobytes of RAM and, once one had scooped a bit of address space for the operating system and a few other essential internals, there wasn't a whole lot left for one's programs to run in.

By comparison, the IBM PC and its later descendants can access up to ten times this memory capacity without any serious sweat and, as we'll see, just a bit more than this with a mild bit of cheating. The PC/AT systems can access still more RAM... sort of. If the designers of the system hadn't been quite as short sighted as they were, it would probably be quite happy with considerably more memory still. However, back in the days when the PC was first spewed forth from the back rooms of IBM, six hundred and forty kilobytes of memory must have seemed like a lot.

By the time the older eight bit computers became something more than hackers' toys, memory was cheap enough that they pretty well all came with all the chips they could carry, and no one really had to worry about things like memory setting switches and segments and so on. The PC, however, because it can hold just about as much RAM as you can afford to stuff into it, has had to make provisions for telling itself what it should expect to see in its allotment of silicon. As such, there is a rather confusing array of switches and jumpers to be conjured over when one is attempting to add memory to a PC.

In this article we're going to check out PC memory in all its glory, from the RAM cards that no one can understand to the DIP switches that make almost no human sense... to some unfathomably simple tricks that will make you the master of every byte in your system, disable the RAM check and allow you to boldly allocate where no man has allocated before.

Beam Me Up, Scotty

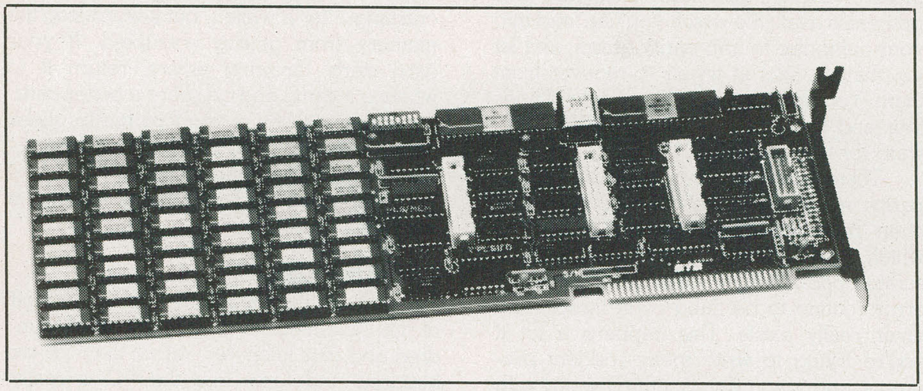
For practical purposes, the memory *range* of the IBM PC stretches from zero to 0FFFFFH bytes. You probably don't know what number five Fs represent either. Don't sweat it. It doesn't matter.

For the purpose of this discussion, just imagine that... as far as the PC's processor is concerned, anyway... there's room for one extremely large chunk of memory in there.

Unfortunately, there are a few things in that space even before we start adding cards and chips that determine how the memory space will be used. The very top of the space is occupied by the IBM BIOS... that's a little bit of permanent program that makes the PC minimally intelligent, giving it enough sense to boot up DOS when it first comes on, as well as handling some basic communications with the real world. You can largely ignore this.

The space at the very bottom of the memory space is occupied by some scratch pad space for the computer itself. We won't

get into just what this space is for just now. Suffice it to say that part of the first bit of memory in the system will get gobbled by the computer, or, to put it another way, there has to be at least enough memory in the system to make this part of things work.



Having it stuffed with enough memory to actually run programs is, as far as it's concerned, optional.

Right smack in the middle of the memory space, in a somewhat stupid location, there is a chunk of memory which represents the video RAM, the memory that holds the characters which are displayed on your tube. This RAM isn't scooped from the computer's system memory or anything so bourgeois... it lives on the video board... but because the location of this memory area is immovably fixed in the middle of the system's address space, one can only have contiguous memory up to the bottom of the video buffer.

The bottom of this buffer sits at seven hundred and four kilobytes. Actually, however, as we'll see, the space between six hundred and forty kilobytes and the buffer is sort of reserved. The designers didn't want people to use it and, as such, only made provisions for switches to allow for six hundred and forty K. We'll be back to this.

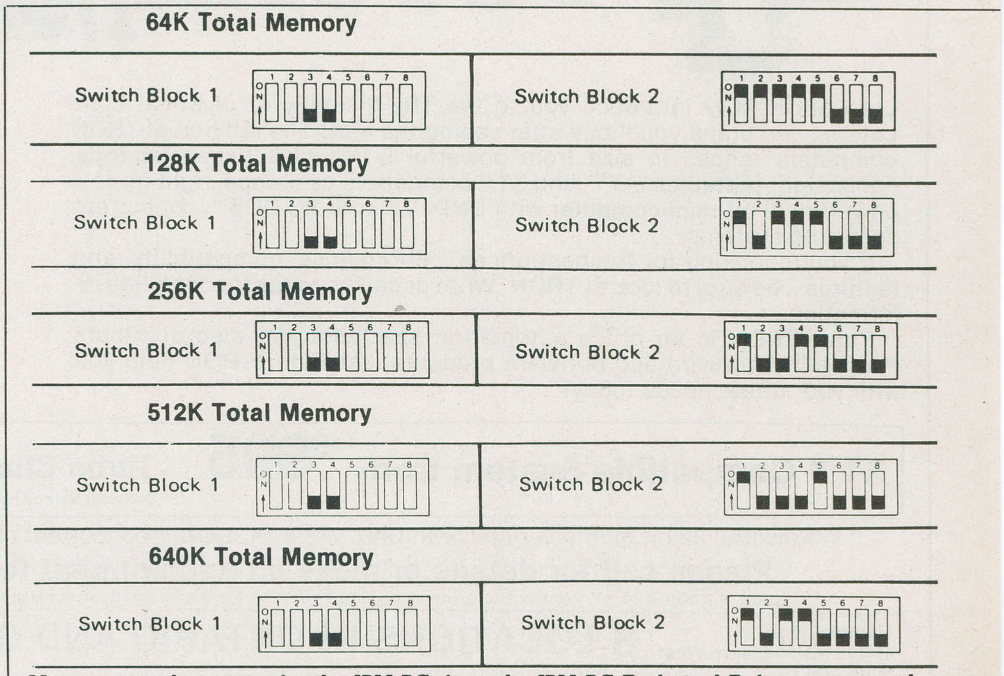
The slots in the PC... into which one plugs memory expansion cards, among other things... are a part of the PC's memory bus. A memory bus is a series of copper tracks to which one can connect memory cards... or, in fact, any memory-like thing... and have them appear as memory to the computer. This bus also extends under the on-board memory chips in the PC.

When everything has been set up properly, memory on the PC's motherboard and memory that lives in cards will all look the same to the computer. It doesn't care where it's located physically, so long as it's in the right places on the bus.

The first question that the astute galactic traveler might pose about this concerns how the PC knows to differentiate between the memory on the motherboard and the memory on a card. For example, if all the memory on the bus is effectively connected together, one might wonder how the computer knows that the card memory starts at the point where the board memory leaves off. In fact, it doesn't. The card knows, however.

Every physical chunk of memory has a base address select mechanism of some sort. You can think of this as a way of telling the chunk of memory on your memory card where to start. If you have two hundred and fifty-six kilobytes of memory on your motherboard you'll want the base address of your card to be two hundred and fifty-six K, as this will make it extend the memory on your motherboard by as many Ks as there are on the card.

Most of the time the base address select thing is a set of DIP switches. If you bought your card from some place that's into heavy packaging and English instructions, it will have no doubt come with a manual that will translate the settings of these switches into actual numbers.



Memory switch settings for the IBM PC, from the IBM PC Technical Reference manual, reproduced with the permission of IBM.

If you have enough slots, you might well have more than one memory card in your system. Assuming that you had two hundred and fifty-six kilobytes on your motherboard and two RAM cards, each with two hundred and fifty-six kilobytes, the first one would be set up to have a base address select of two hundred and fifty-six K. The next one would be set up for a base address select of five hundred and twelve K. Actually, there's a problem here... the upper card would have part of its memory overlapping the video buffer, a very bad thing to do.

Most memory cards also have bank enable switches. These will let you disable part of the memory on your card, that is, make it appear to the computer as if it doesn't exist. They're used to overcome problems like this one.

Assuming that your card did come with an intelligible manual, getting the switches set up shouldn't be too heavy a sweat. You can have memory ranging up to seven hundred and four kilobytes, or 0B0000H in computer terms, without upsetting anything.

The tricky bit usually comes in setting up the PC's memory size switches. Because the system can have varying amounts of memory in it, the PC must be told how much RAM to expect. This is gotten together with the memory selection switches. They turn up in varying locations amongst the motherboards of varying PC compatibles, although they all work the same way. The chart around here somewhere illustrates some of the common switch settings to use. There are hoards of intermediate values, but they're rarely applicable to the sorts of memory cards one comes across at the moment.

You can set the memory switches to values representing too little memory without any nasty effects... beyond your

computer simply having less useful memory to work with. If you lie to your system and tell it that there is more memory than there really is, your computer will be very unhappy upon booting, as it will think that the memory that doesn't exist is actually bad memory. Various BIOSs handle this sort of error in various ways.

Once you have your system booted, you can run the CHKDSK utility that comes with DOS to see if the amount of memory the system thinks it has is about the same as what you set the switches to. If the two values don't agree, you've probably gorched the switches.

There are a few other bits of basic memory lore to be concerned with. The most notable of these is the occasional instance of parity errors, or parity checks. If you've ever encountered one of these things, you'll know that the usual result is to hang your system unexpectedly, trashing everything you had going at the time of the great crash.

Each "byte" of memory in the PC is actually nine bits wide, the highest bit being a "parity" bit. This is a sort of flag that allows the PC to check the byte for corrupted data. Its response to finding corrupted data is to crash itself. This seems a bit drastic to me, too, but that's the way of it.

Some of the compatible BIOSs allow you to choose either restarting the computer or trying to get back into your application in the event of a parity error. This is notably more civilized, to be sure. There are also programs available which you can load into your computer to trap parity errors, and let you have a shot at trying to recover from them. You might want to check out volumes four and five of our Almost Free PC Software for these things.

There are a number of causes of the parity error swan dive. One of these isn't even related to parity errors. Programs which inadvertently trample on the PC's scratch pad memory may cause the parity error routine to be flung even through no error really exists. This happens a lot if you're trying to write tricky resident programs, and if you do something weird with resident utilities like SideKick. Combining a lot of these things that aren't all mutually compatible can have odd results like this.

Bad memory is a meaningful cause of parity check errors and, in fact, it's what the whole system is largely designed to trap. Parity errors will often turn up if you have a funky memory expansion card. To be sure, if you lay a new board on your system and these little trolls start popping up you should suspect the card.

Some of the less well designed cards have mechanisms on them to disable the parity checking of the PC. This is a very bad trip. It will stop your computer from crashing... for a while... but it won't stop the memory from getting corrupted. If your card starts throwing errors, return it to whence it came and go score a better card.

Other causes of random parity check errors are dirty power, a funky power supply, static shocks, random cosmic rays, bad slot connectors and phantasms lurking in your disk drives.

Recollections

I mentioned a while ago that the upper limit of the PC's memory is actually seven hundred and four kilobytes, rather than the six hundred and forty that the PC is really prepared to let you stuff in there. The sity-four kilobytes in between is a kind of grey area which has been reserved by IBM for future expansion, but doesn't serve any function on most systems at the moment. You can get memory cards which can place RAM into this space... it will usually mean banking out some of the memory on a larger card... and, if you could get the

system to recognize its existence this space would be useful for things that gobble a lot of RAM, such as Lotus or Personal Composer.

The way to get this together is to use a little program called RAMSET.COM. It lives in your AUTOEXEC file, and runs... twice... when you boot your system. The first time it runs it actually checks each chunk of memory starting from where it is up until it finds some that isn't there or until it hits the base of the video buffer. Then it puts the address of the *real* top of the memory into the part of the PC's scratch pad memory that holds its memory size. Finally, it reboots the system.

On the second booting the PC's BIOS foregoes its memory check, relying on the value in its scratch pad which it thinks it set the first time it booted. The RAMSET program runs a second time but, finding no discrepancy between the memory in the system and the value in the PC's scratchpad, it doesn't change anything and sees no good reason to go on rebooting. As such, it drops into DOS as the system normally would have done on the first boot.

The result of this is that the system will think it has as much memory as is actually plugged into it, rather than the amount the switches tell it is there. This allows you have memory up to seven hundred and four kilobytes even through the switches don't allow it, and it also allows you to bypass most of the wait for the power up memory check imposed by older BIOSs that don't provide for a way to kill this from the keyboard. You can set the switches to something very small... say, sixty-four kilobytes... and let the RAMSET program adjust the memory to its true value when DOS boots up. The sixty-four kilobytes is just enough to run DOS in.

The RAMSET program is actually a tiny assembly language routine, but, to make it easy to use I've created a BASIC loader for it. Type the little program in listing one into BASIC, save it and run it. It will produce a file called RAMSET.COM. If it comes up and gives you a checksum message you've committed some sort of ghastly typographical error in the data statements.

The RAMSET.COM file the BASIC loader creates is a runnable program... you don't actually need the BASIC code after it has been created. Put it in the root directory of your disk and put its name in the AUTOEXEC.BAT file... preferably as the first thing, as it will trash everything before it on the first booting anyway. You can now set the switches of your PC really low and still have access to all the RAM in the system.

It's probably also worth speaking about RAM disks. Because the PC has so much memory available to it, it's quite practical to block off some of this and make it act like a disk drive. This may take a second to get used to if you haven't encountered it before. With a RAM disk in place, the computer creates, for example, a fictional drive C on a two drive system. If you write to drive C, the information will be stored in a chunk of protected memory. You can do almost anything to drive C you would do to a floppy.

```
10 ' ramset driver (c) copyright 1986 Steve Rimmer
20 ' creates RAMSET.COM. Place RAMSET.COM in your
30 ' autoexec file. Sets PC memory settings and
40 ' reboots the system with a weird message.
50 '
60 CHECKSUM = 10524
70 OPEN "O",#1,"RAMSET.COM"
80 FOR X = 1 TO 101
90 READ A : PRINT #1,CHR$(A); : C = C + A
100 NEXT X
110 CLOSE
120 IF C = CHECKSUM THEN END
130 KILL "RAMSET.COM"
140 PRINT "CHECKSUM ERROR"
150 END
160 DATA &H1E,&H33,&HC0,&HE6,&HA0,&HA1,&H02,&H00
170 DATA &HBB,&H00,&H00,&H3D,&H00,&HB0,&H74,&H1A
180 DATA &H8E,&HD8,&H89,&H07,&H8B,&H17,&H3B,&HC2
190 DATA &H75,&H10,&HB9,&H08,&H00,&H8E,&HC0,&H33
200 DATA &HC0,&H33,&HFF,&HF3,&HAB,&H8C,&HD8,&H40
210 DATA &HEB,&HDE,&H50,&HB0,&H80,&HE6,&HA0,&HB8
220 DATA &H40,&H00,&H8E,&HD8,&HBB,&H13,&H00,&H58
230 DATA &HB1,&H06,&HD3,&HE8,&H3B,&H07,&H74,&H0C
240 DATA &H89,&H07,&H1F,&HBA,&H4F,&H01,&HB4,&H09
250 DATA &HCD,&H21,&HCD,&H19,&H1F,&HCD,&H20,&H47
260 DATA &H6F,&H69,&H6E,&H67,&H20,&H74,&H6F,&H20
270 DATA &H68,&H65,&H6C,&H6C,&H2E,&H2E,&H2E,&H20
280 DATA &H77,&H61,&H69,&H74,&H24
```

The BASIC loader for RAMSET.

The advantages of a RAM disk are in that it's extremely fast. If you put something like WordStar on it, WordStar will clip along at speeds that far exceed those of even a really good hard drive. Second, a RAM disk can give you more on line space when you've got both your floppies tied up.

There are numerous practical uses for RAM disks. In writing large programs in C, for example, I boot the system with a disk that creates a RAM disk and puts the editor and compiler onto it. Then I put the two disks with the source code and other files in the drives. This means that the editor and compiler run very much more quickly than they could have otherwise done from floppies. They also don't steal any space on my working disks, which would otherwise seriously limit the size of the programs I could work on. Finally, the compiler can write its scratch files and temporaries to the RAM disk, which speeds it up still further, obviates the need to delete the scrap and further reduces the system's requirements for disk overhead.

The drag about RAM disks, of course, is that their information is extremely volatile. If your system crashes, the contents of the RAM disk evaporate. RAM disks are good places to put programs so they'll run faster... but they aren't very good homes for source files, documents or other information that would have to be rekeyed if it were to be lost.

The software that makes a RAM disk happen usually exists in the form of a DOS device driver. I won't get into how one of these things works just now. Suffice it to say that you can get it together by putting the RAM disk program file in your root directory and placing the line

DEVICE = RAMDISK.SYS

in the CONFIG.SYS file on your disk. The file name... RAMDISK.SYS here... should agree with the name of your RAM disk program.

If you're using DOS two, you'll have to acquire the RAM disk software from somewhere. It's available from us on Almost Free PC Software volume four. There's also an assembly language source file for a RAM disk program on volume six, should you be just dying to know how they work. If you have DOS three, a suitable RAM disk, VDISK.SYS, is included with it.

IBM PC Memory Shuffle

Theoretically, RAM disks shouldn't interfere with other programs running on your system, except for swiping some memory for their own uses. In the real and funky universe that they find themselves stuck with, however, they can do some peculiar things. This is especially true of some of the less well written ones, and of some of the more peculiar software, which may overwrite the memory that's supposed to be reserved by the RAM disk and corrupt it. As with any DOS based disk, if you almost fill a RAM disk and try to write to it, it can get extremely slow even if it doesn't throw a full disk error. Programs which use virtual memory... passing parts of large files to and from the disk when they run out of real memory... may also be slowed down by a loaded RAM disk, both because it has scooped some of their RAM and because it's slowing down upon getting stuffed.

If you decide to use a RAM disk that has your applications loaded into it when your computer boots... for instance, WordStar... there are a number of things you might want to check out. For instance, you can set up a path into the disk to make your programs easier to use. Including this line in your AUTOEXEC file,

```
PATH C:
```

will allow you to type the name of any program in the RAM disk while you're logged onto one of your floppies and have it run as if it were on the floppy disk. The catch in this, however, is that programs such as WordStar, which look for overlays, will have to know to search for them on drive C, rather than on the drive you're logged onto. You might want to look at the WordStar patching article elsewhere in this edition of Computing Now! to see how this can be gotten together.

Other Memories

There are a few advanced bits of memory flotsam that are worth mentioning in conjunction with the PC. The most notable of these is the idea of bank switching, a process which can allow really memory hungry programs running on the system to access more RAM. The space between the top of the video buffer and the bottom of the BIOS isn't used for anything in the normal scheme of things. However, it's possible to have a suitably sophisticated bit of hardware make pages of memory show up in there. Under bank switching systems, huge chunks of memory... several megabytes, if you can afford it... can be accessed by moving small chunks of them in and out of this address space window. The most common manifestation of this is the Lotus Intel bank switching standard, which was designed to run with Lotus 1-2-3.

There are a lot of catches to this. The first one is that this extra memory doesn't turn up as useable program memory at all... it can only be accessed by specially written applications programs that are specifically designed to run the bank switching mechanism that will pop the pages of bank switch memory in and out of the high memory window on the PC. Few programs need this much storage, so there aren't very many applications which use this technique beyond Lotus itself. The appropriate memory cards are ruinously expensive.

One might argue, though, that anyone who is running a company big enough to need a spreadsheet that won't fit into almost three quarters of a megabyte of memory can probably afford any card in creation.

The second catch is one of speed. It takes hundreds of times longer to bank in a page of memory to read or write to it than it does to simply address a location of regular program memory. This can be further complicated by a phenomena called "thrashing". If the program wants to repeatedly address locations of the banked switched memory which lie in different pages, it will have to flip the pages in and out of the address space window quite a lot... which can really get the whole dog and pony show down to a slow trudge.

If you're up in a higher tax bracket, you can probably get around any serious memory crunches by buying a PC/AT, the most recent offering from the chip freaks. While this has program memory that behaves in the same way as does that of a regular AT, it also has extra address space that allows for several megabytes of extended memory. This space can't be used as regular program space, but it can be used to store large data arrays relatively easily. It will also serve as the RAM disk of the gods. The memory arrangement of the AT solves most of the limitations of the PC... pity it's too expensive for most of us.

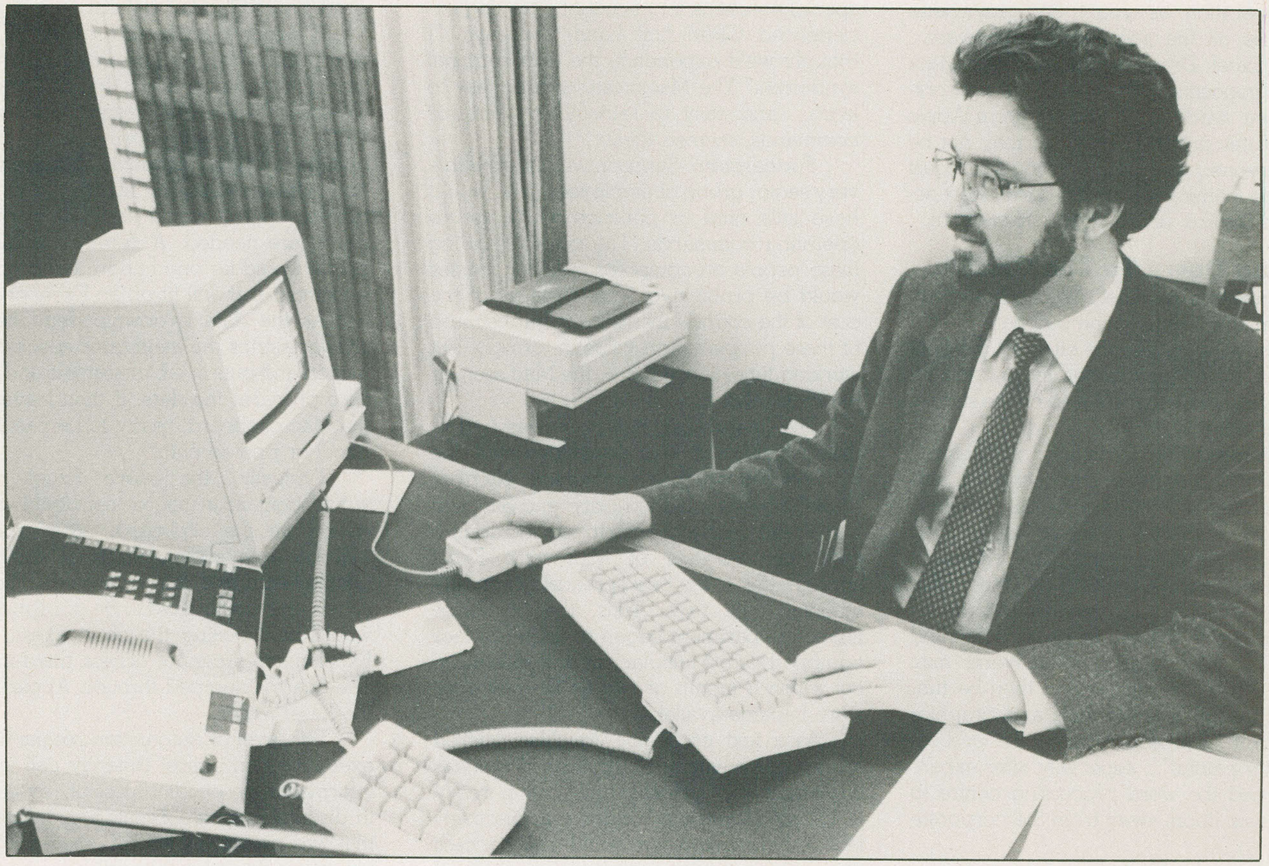
Fundamental Karma

There is probably a fair bit more that should be said about the memory in the PC... I'll leave it for another article, though. This one is fairly huge as it is.

Perhaps the most interesting thing about the PC's memory is how all the software for the system has grown to fit it. It's enlightening to note that every program ever written for eight bit computers ran in sixty-four kilobytes and enjoyed it. I can't think of much that will even boot into sixty-four K on a PC, let alone do anything practical. Typically, the more interesting PC programs need at least a quarter of a megabyte.

Of course, one didn't have things like memory resident note pads, background task Zaxxon games and so on old CP/M based machines. All they did was to run their programs, crash the odd disk and breath the clean fresh air of simplicity. Their like will not be seen again.

Except, perhaps, as bookends. They were always good for that. **CN!**



The Perfect Macintosh

Apple dreams of businesses ordering truckloads of Macs. While the reality to accompany may be slow in coming, here's a look at one company that has put mice on the desks of their employees.

by Frank Lenk

Many an industry prognosticator and advertising agency has considered the great question, whether the Macintosh can survive as a business system. Apple, naturally enough, says that it can. Many others say only maybe... a vicious few say that it hasn't a chance. The real story is that all these statements probably make a great deal of sense, providing you look at them from the right point of view. One man's Macintosh is another man's massacre, so to speak.

After delving more deeply into the matter, we eventually unearthed what has to be *the* perfect business implementation of the Macintosh. Apple itself provided the first clue... relaying me down the line to SES Computing, a dealership located deep in the heart of downtown Toronto's concrete canyonland. SES installs vast numbers of business Macs, and was therefore able to refer me in turn to several potentially enlightening case histories. I corralled one of the most impressive of these corporate cases, McLeod Young Weir.

McLeod Young Weir... simply "McLeod" to their friends... occupies six or seven floors... there seemed to be some doubt as to the current status... at the top end of the TD Centre's commercial union tower. The company is one of the most massive investment houses anywhere. It "provides its clients with a complete range of domestic and international investment banking, brokerage, trading and financial advisory services", or so it says. There are no fewer than forty McLeod offices across Canada, as well as branches in New York, London, Zurich and Tokyo. However, head office is right here in Toronto. According to their own prospectus, "McLeod has traditionally been ranked as one of the top two investment banking firms in Canada..." This is truly the big time.

I spoke with one Thomas H. Simpson, of corporate finance. This is the real core of the outfit, with a professional staff of over forty shrewd brains. Twenty-two members of this department are directors of the firm... the real top guns. These are the guys who are expected to have their fingers on the pulse of the commercial world. Aside from any skills in this area, however, Simpson himself is the self confessed instigator of the entire move into computers. It was he who made the somewhat courageous decision to forge ahead with the Macintosh. You might as well know up front that the story has a happy ending. The decision turned out to be an inspired one. An infusion of Macs has affected the company the way a shot of steroids affects an athlete.

McLeod now has about as complete a

Mac operation as you'd find anywhere. They use all the popular Mac software... Word, Excel, Draw and so on. Their Macs are all connected via AppleTalk networking. Their hardcopy all comes from Apple LaserWriter printers. The Mac may not be for everyone... but it's interesting to see just what the perfect Macintosh installation actually looks like, and how it comes about.

Getting Down To The Crunch

As an investment dealer, McLeod must advise its many clients on the purchase of new stock issues, elucidate the effect of mergers and acquisitions and generally provide insightful financial advice based on the state of the economic world.

This means lots of number crunching... analyzing, forecasting, evaluating. It also means communicating the results... spewing forth vast quantities of paper embellished with words and numbers. The output of finished paperwork prior to the Mac invasion had been entirely centralized. Word processing was something that was handled entirely in one centre, on a Wang mini system. Processed text would then be funnelled through a typesetting and graphics department. Simpson recalled the process as "pretty painful"... especially after expansion forced the word processing centre to relocate six floors away from the corporate offices.

Draft text at that time was originated as scrawled manuscript... a barbaric practice, you'll agree. Forcing these hand written scraps of verbiage through the machinery was a complex problem in logistics. Early drafts would come back from word processing "with a billion typos", needing tedious correcting and multiple passes through the whole dog and pony drill. The lack of efficiency cut into quality, since there would never be enough time to get things totally polished up.

This anemic state of affairs was obviously in need of a good shot of silicon. "I was the prime mover behind it all," says Simpson.

Simpson had already owned an Osborne system. He'd lugged it in to work occasionally to do the odd bit of numeric modelling, but found this a cumbersome way of working. The somewhat autocratic McLeod computing department "had a stranglehold on computer resources"... a state of affairs still woefully common in the corporate world, by all accounts. The standing policy was to go mainframe first, and if you absolutely *must* have a micro, it had better have a blue, three letter logo on it. Like Simpson, some of the junior people joining the company would already have their own machines, but that was as far as it went.

When Simpson started looking at equipping the corporate finance department with micros, he had to take some special concerns into account. "Having experienced CP/M first hand," he says, "I thought we'd experience the dark terminal syndrome." The Macintosh, with its easy to learn... and easy to remember... graphic commands, seemed ideal.

Furthermore, Simpson was faced with a very senior group of employees, who would have little time or patience with obscure operating commands. The cost of sending these people on computer training courses would be prohibitive... not because of the cost of the courses themselves, but because of these people's high value in terms of lost working time. Once more, the Mac seemed the only solution.

The final factor, however, was performance. Simpson was convinced that the Mac offered superior technology. Number crunching was a high priority, and Simpson had already had a chance to see and admire beta test versions of Microsoft's Excel spreadsheet... "the best spreadsheet on any machine," he says. Thus, the corporate finance department began to acquire Macs equipped with Multiplan, upgrading to Excel within a month or two as the new software became available.

As a text processor, the Mac offered some clear advantages. Its ease of use was important, but seamless integration really made the difference. Under a single interface, even casual users could easily access disparate applications such as Excel, Word or MacDraw.

Simpson took his proposal through the official channels. The corporate systems committee rejected the idea of going Macintosh, but the executive committee decided to go ahead anyway. Since then the computing resources crowd have seemingly turned the corner. The *new* department head is more open minded, and admits that PCs and Macs have a proper place in the total computing picture.

Today, the one application still not handled on the micros is data base management. All of the company's data files continue to reside on a DEC 2060 mainframe. These files include such things as a complete trading history of all stocks in North America, including information such as daily maximum, minimum and closing values. There's the Financial Post database, with company balance sheets and other data going back ten years. There are new issues, stocks, bonds and other trading data. You can see why it all needs a mainframe.

Prior to the advent of the micros, all this data was accessed only by a corporate specialist... usually an MBA graduate who had to be specially taught APL in order to deal with the mainframe system. Once the Macs were available the company was able to acquire PortaAPL and create its own terminal program to access this massive information bank from every desktop.

The connection is via a twenty-four hundred baud Gandalf dataset line. The PortaAPL program loads APL on the DEC and gets the user's password protected workspace. A user might, for instance, need to access the stock time series database. The DEC software would ask him for the exchange and the product's ticker symbol, and whether daily or monthly data is required. Then it would ask how many data points are needed. A common number of points would be one hundred and one, the maximum that will fit on an Excel chart.

As the data is downloaded, the Mac software strips the mainframe specific information and puts in tab characters and otherwise prepares the data. It then leaves it on the Mac clipboard, ready to be pasted into Excel or printed out.

Typically, the user's Excel system would contain a macro that could get the data from the clipboard, paste it into a template area on the spreadsheet, do a bit of manipulation and display a chart. The only action that would have to be done manually would be labelling the chart axes... since Excel macros cannot access this function. The chart is dumped through AppleTalk to the LaserWriter.

This level of automation means that an executive can almost instantly call up any information he needs... possibly even while in discussion with a client. Mainframe data that was previously all but inaccessible has become a fingertip resource.

APL was used on the Mac for two reasons. It interfaces well with the DEC, and it was the only language in which the company already had expertise. A system programmer is being hired, and one of his functions will be to "Mac-ify" the terminal program's user interface, making it even more natural to use.

The first Macs came in last summer. They landed primarily on junior executives' desks, to be used for number crunching. Gradually the senior staff acquired their own hardware. The last to go was the secretarial level... saying something about the type of operation McLeod was getting into.

The Hard Facts

The top floor of the TD tower now houses two complete AppleTalk networks, each one with its own one hundred and ten megabyte Sunol hard disk and LaserWriter Plus printer. At the time of my visit, one network had sixteen Macs, the other fourteen. Both were creeping up toward the twenty Mac mark.

McLeod corporate finance has branch offices in Montreal, Vancouver and Calgary. Each now has its own network of Macs. These mimic the Toronto installation except that the Calgary office is using a Keeper brand hard disk, with twenty megabytes of fixed storage and a ten

megabyte removable cartridge. This drive is both lower in capacity and slower than the Sunol drives used in Toronto and Montreal, but it suffices to deal with the five Macs in the Calgary office.

Since the Mac systems went in, Simpson has made several discoveries about the hardware.

For one thing, all the original Macs were equipped with ImageWriter dot matrix printers. The idea was that drafts could be done locally, leaving the LaserWriters free to handle a greater flow of clean copy. The ImageWriters now function solely as dust catchers, for two reasons. First of all, the LaserWriters have proven more than capable of handling all the printing the network can throw at them. Second, it turns out that Microsoft Word has a slight difference in the way it treats formatting on the dot matrix and laser printers. This makes switching back and forth more trouble than it's worth.

The AppleTalk network has presented only one real hardware problem. Says Simpson, "The biggest problem we've had is people kicking the network apart." Apple touts the network as being easy to plug together. Apparently it comes apart just as easily, particularly where the junction boxes lie around underfoot. The McLeod solution has been to encase the connections with heat shrink tubing. This stuff goes on easy and can be quickly cut apart if things need to be moved around.

Hooking the Gandalf communications system up to the new Mac Plus proved to be trickier than it looked. Apple eliminated the five volt output line from the new modem port. Since the Gandalf uses this power, SES Computing is having to modify the Mac Plus ports on McLeod's new machines back to the original Mac standard.

The Sunol hard disk was apparently one of the first network servers available for the Mac... even before Apple had AppleTalk available. The one hundred and ten megabyte drive splits into seven virtual drives, each containing six to ten password protected volumes. Each user has his or her own three megabyte applications software volume plus a file volume or two. Even so, there's lots of space left.

Sunol volumes are installed through a special desk accessory, *Sunol Mount*. Once installed, each volume appears on the desktop just like a normal Mac disk icon, and can be opened in the usual way. The hard disk system runs SunTalk 1.0. Version 2.0 is apparently available, and supports Apple's new hierarchical file system. Its response... particularly with the new, improved Mac Plus... is said to be quite snappy. It's not quite as good as having your own local hard drive, but a whole lot better than working from floppies. Using Apple's mini-finder helps speed things up even more.

Soft In The Head

Although numbers are at the root of the business, most of the computing time at McLeod Young Weir is undoubtedly soaked up by text processing. Thus, Microsoft Word is probably the number one application.

Simpson says "We're not finding any limitations." Of course, there have been the inevitable teething troubles. For instance, it took quite a while to track down the program's double underline function. Otherwise, Word has provided complete satisfaction.

Excel, as I mentioned above, has been even more satisfactory... if that's possible. It functions as the mainstay in number mashing.

MacTerminal has been used for accessing any dial up data services. However, something called *Q&D Terminal* is being evaluated, and will probably take over since it runs as a desk accessory, available at all times.

MacDraw is used to enhance Excel graphs, generate transparencies and do the occasional organizational chart. Simpson envisions heavier use of this program in the future, once everyone gets better oriented toward communicating ideas in a graphic manner.

Some software incompatibilities have come up. Simpson was using ThinkTank, for instance, but found it wouldn't work right on the Mac Plus. An update from Living Videotext fixes the problem. Far stranger is the incompatibility between Apple's own Switcher and the AppleTalk network. Attempting to run several programs concurrently under the Switcher can cause a crash in certain circumstances. Simpson was not enthusiastic about the Switcher concept in any case. Running multiple applications at one time places an awful lot of data in jeopardy, and crashes... however rare... are an inescapable fact of life.

Another software problem cropped up because the Sunol system doesn't support certain desktop functions. For instance, dragging a disk icon to copy it won't work properly with the Sunol's odd sized volumes. The original desktop operation is based on the assumption of uniform volume size. More drastic consequences will result from any attempt to use the pull down *erase* function on the disk menu. To avoid accidental decimation of the hard drive, this function has been hidden away on all the McLeod systems. Using the Apple resource editor the pulldown menu has been altered so that it no longer shows the erase option at all. To be absolutely sure, the subsequent dialogue box has also been edited to say something like "You *must* choose cancel" This seems a simple yet useful trick, and could probably be applied in all sorts of circumstances.

Simpson has managed to find one serious gap in the Mac's software coverage. Although he's tested two or three packages, he's been unable to come up with a "half decent" spelling checker. Of the ones he's looked at, some have been either too slow or too incompatible with AppleTalk. Others have contained American spellings or other faults in the dictionary that could not be edited out.

To Mac Or Not To Mac

Simpson's report may sound like a suspiciously glowing testimonial. To inject some perspective, let's just examine some of the qualities that made McLeod such a fertile ground for planting apple seeds.

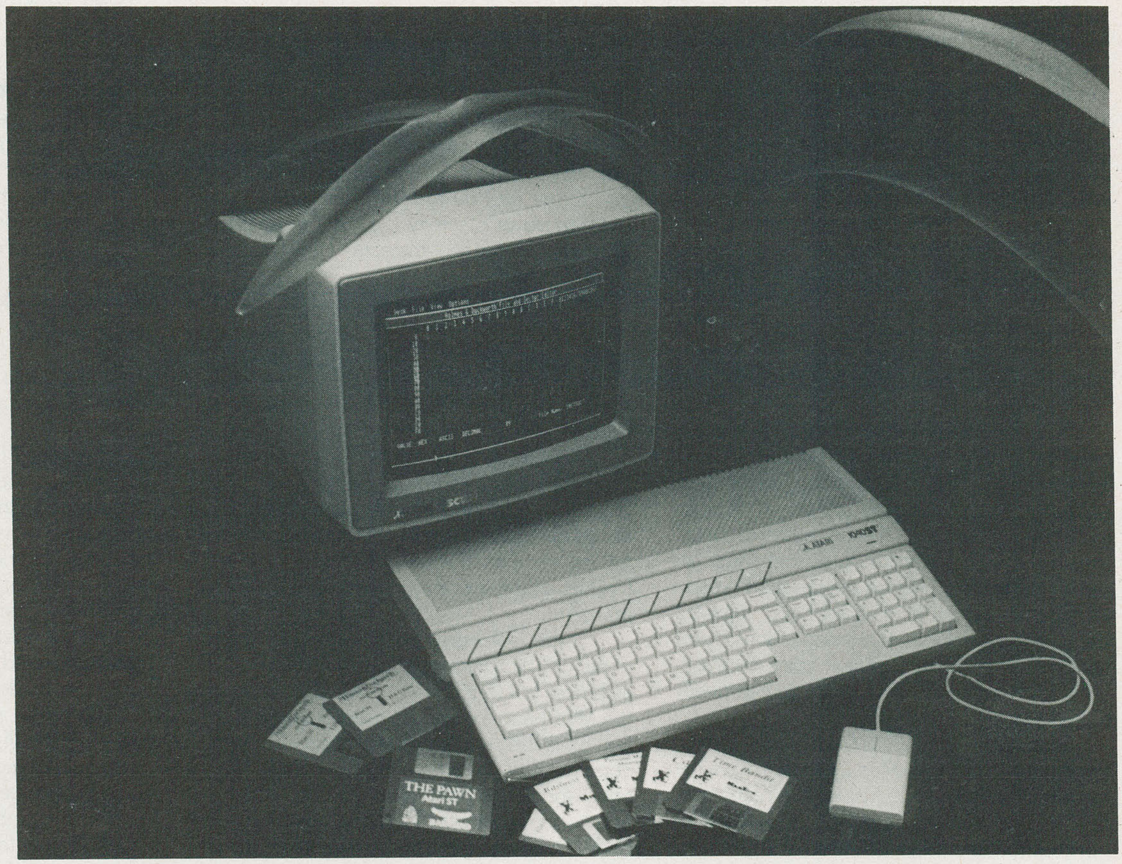
First, there was the complete computer vacuum that Simpson started with. There was no need to fit into previous conceptions of any kind. Second, there was a high power, very senior management type of staff... people who would have had little previous computing experience and even less patience with esoteric computing environments. They would also have little prior expectations as to brute force performance. For instance, none of them would be likely to have benchmarked a PC with math co-processor versus a Mac. These were busy executives who had earned the right to be pampered.

Third, there was a limited range of software required... and that range was almost a tailor made fit for the Mac. McLeod, Young Weir needed a top flight spreadsheet... Excel... a powerful, easy to use word processor... Word... plus some odds and ends available on darn near anything except maybe an Intellivision game console... a terminal program, a graphics package and whatnot. They needed no great selection beyond one decent package in each category, since the systems were essentially to be used by computing dilettantes... people who would skip lightly from one application to another, without pushing any of them to the uttermost limits of performance.

Finally... right in line with Apple's own emphasis on the desktop publishing diagonal market... McLeod needed to produce a mixture of graphic and text material quickly and painlessly. This is a circumstance that really brings out the best in the Apple's standardized environment, and really lets the LaserWriter shine.

What McLeod did *not* need was flexibility, compatibility, or brute computing power. There were no peculiar devices to interface onto the bus. There was no need for fancy colour displays. There's only one custom tailored piece of software in use at the company, and no ongoing need for any further programming reach. Therefore, there is also no need for the advanced functions provided by a Unix type operating system, such as I/O redirection or shell programming. **CN!**

Software for the Atari ST



While hardly inundated with software just yet, the Atari ST is starting to accrue a library of sorts. The state of the art is outlined in this feature.

by Frank Lenk

A computer is only as good as the software you can run on it. Unless you feel up to creating all your own applications... in which case you might as well build your own dream computer, as well... you'll be wanting to see a selection of commercial software ready to run on your chosen hardware.

When the Atari ST was introduced, it looked like a cute machine, but one immediately wondered what it could *do*. The answer was, very little. There was no software.

That situation has changed. The selection is varied, with some excellent packages in most areas. It's not exactly rivalling the IBM PC population, but it's amazing how many programmers out there... especially in the UK and Europe... have had a hankering to program a box with a 68000 CPU inside. The English have been getting by on the Sinclair QL for some time, which should give some idea of just how strong these yearnings have been.

The floodgates are open. As the rising waters eddy about our knees... metaphorically speaking... I've snatched a few samples from the current. What follows is a more or less informed selection of what's going by.

The Game's Afoot

The real good news is in games. The ST is just now beginning to fulfil some of its theoretical promise as a games machine. I know Atari will have mixed feelings about any praise in this area... they'd desperately like the ST to be seen as an all round home and business do everything sort of computer. Alas, this image is in dire jeopardy, as top notch games proliferate.

There are a lot of old standards being ported over to the ST. Not surprisingly, these are no more entertaining in sixteen colours than they were in two or three. A prime culprit in this regard is Penguin, with its Polarware series. These efforts... like Transylvania or Sword of Kadash... may have seemed like the last word on the Apple II, but they look a bit antiquated on the ST. Actually, the ST probably makes them look worse than they really are... as games...

To Over 30,000 Installations, MultiLink® MEANS Multi-User.

At over 30,000 sites, as many as one-quarter of a million users tap into the power of MultiLink® Advanced everyday.

Since 1981, they've come to rely on our multi-tasking, multi-user system for compatibility with their favorite software, and the ability to share disks, files, printers, and programs in a *true* PC-DOS environment.

From the largest of the Fortune 500 to the smallest in small business, MultiLink® has provided a cost-effective multi-user solution that's available from no one else.

MultiLink® Means Cost-Effective Timesharing on a PC. MultiLink® Advanced utilizes the principle of timesharing by sharing a central PC's peripherals, files, and processor time among nine users. Up to eight inexpensive terminals can be connected to a single non-dedicated IBM PC, XT, AT or 100% compatible using standard RS-232 ports. Each terminal effectively emulates a PC having up to 512K RAM.

PC-Shadow™ Workstations, shown below, even have an AT look-alike, as well as work-alike, keyboard, display, and serial port. In addition, password-protected remote access via modem can be made with either dumb terminals or PCs running our terminal emulation software.

MultiLink® Means PC-DOS Compatibility with a *Software-Driven* System. Lotus 1-2-3, Symphony, WordStar, dBASE III, & Multimate are just a sampling of the wide variety of PC-DOS software that's fully compatible.

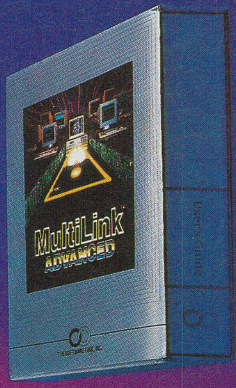
Our software-driven system is also IBM NETBIOS compatible, so programs that are written for IBM's Token Ring will run on our multi-user system, as well.

MultiLink® Means Multi-User to Leading Computer Publications. Whether you read *PC Magazine,* "MultiLink® Advanced delivers on...convenience, speed, and flexibility," or *InfoWorld,* "If you want a low-cost multiuser system with up to eight terminals, MultiLink® Advanced is worth a serious look," it becomes clear that MultiLink® Advanced is a formidable contender in the multi-user marketplace.

See What MultiLink® Can Mean to You. Learn, firsthand, how our multi-user system can benefit your company. Call The Software Link TODAY for complete information and the authorized dealer nearest you. MultiLink® Advanced is $745 and comes with a money-back guarantee.

MultiLink® is a registered trademark of The Software Link, Inc. MultiLink® Advanced & PC-Shadow™ are trademarks of The Software Link, Inc. IBM PC, XT, AT, PC-DOS, Token Ring, & NETBIOS are trademarks of IBM Corp. WordStar, dBASE III, Multimate, Lotus 1-2-3 & Symphony are trademarks of MicroPro, Ashton-Tate, Multimate International, & Lotus Development Corp., respectively.

MultiLink® ADVANCED



THE SOFTWARE LINK, INC.

250 Cochrane Drive, Suite 12 Markham, Ontario L3R 6B7

CALL: 416/477-5480

Dealer Inquiries Invited

Circle No. 6 on Reader Service Card

simply because the scratch built ST software looks so much classier.

Penguin's Transylvania, for instance, is a totally straight forward port of the familiar game already seen on virtually every micro environment in creation. This is a decent game for the kiddies... or for one of those idle afternoons when you feel a bit childish yourself. The graphics look nice on the ST's tube, but can't escape that "two year old with a crayon" look that points to origins on machines that simply couldn't do much in the way of realistic artwork. The plot is not bad, as these things go, but hardly on the Infocom level as far as depth or complexity.

Kadash looks about the same, being burdened with unappealing, blocky graphics and a muddy choice of colours. The title screen, oddly enough, is a really striking desert scene, done in uncannily realistic colour. The actual play, however, takes place on a black field populated by relatively crude images that look like what might be built with some multicoloured Lego bricks. The mouse control is so dodgy as to be virtually useless. I found the keyboard far preferable. I wasn't equipped to try out the joystick option, but that's probably the way to go. However, that doesn't excuse the game crashing on me while I was trying to switch control from the mouse to the keyboard. I *think* I hit the wrong control key combination.

On the other hand, as maze games go Kadash does not seem to be without some interest. You get to collect weapons, build hit points and kill all sorts of nasty critters. If this sounds like your cup of blood, you might want to try it out... preferably *before* you elect to own it.

In marked contrast are the games now appearing under the MichTron banner. MichTron is a company that specializes entirely in ST software. It isn't surprising, therefore, to find that their material attains far better utilization of the goodies that make the ST what it is.

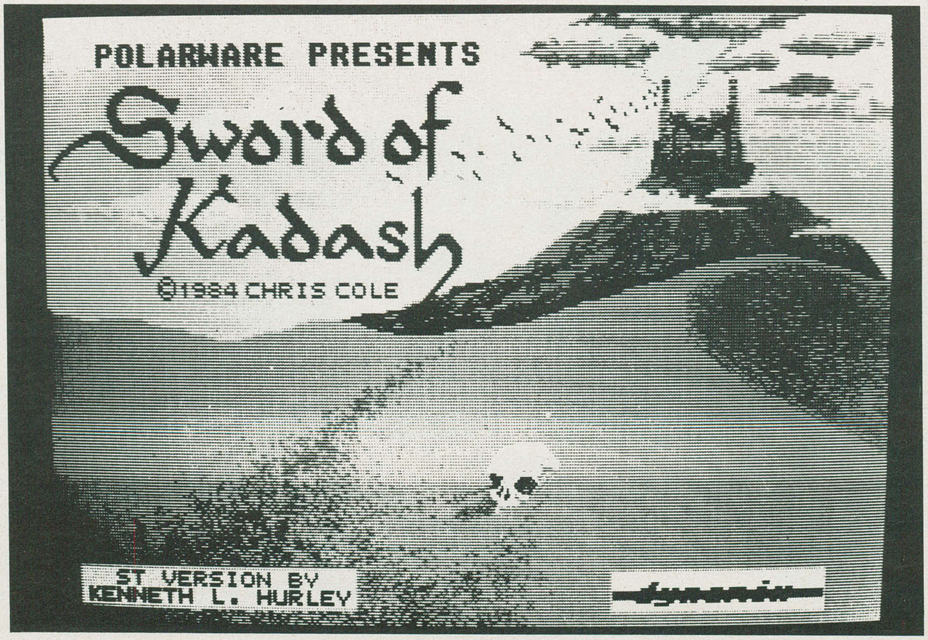
Of the games I've seen so far, MichTron's Time Bandit is definitely the standout. It's really a true arcade quality display on a home machine. The basis of the game is traditional: you are a little man who scurries around a scrolling landscape shooting little animated enemies. As with many other such games, you shoot in the direction you are facing. With a joystick you can stand still and spin around, firing a destructive hail of bullets around you. Though only about a half inch high, your little figure as well as the enemies, have clearly distinguishable personalities. They have features, armament... even little highlights on the glossy bits.

Apart from being detailed, Time Bandit is also *huge*. You start on an introductory field, dotted with houses, pyramids, castles and space ships. Each of these contains a complete game world. Entering any of them throws you into a unique maze populated with its own distinctive variety of deadly perils. Each game has four levels, and each level has a further four sublevels. There are adventure game elements woven in, so that as you attain successively higher levels you also discover new clues to the final puzzle you'll have to solve to complete each game area.

The game's scoring is equally sophisticated. As you progress, an indicator on the screen tells you how brave you are. Steering clear of enemies while maintaining a continuous stream of bullets will rate you as timid. Diving madly at everything that moves will get you known as daring, or even psychotic. Whatever you destroy, your score will depend on your bravado rating of the instant. Gutsy players get more points per kill, but may live shorter lives. Daily and all time high scores are recorded.

MichTron also scores well with some of its less spectacular games. Its Cards package contains remarkably pleasant adaptations of the old favourites. The selection includes blackjack, cribbage, klondike, poker, squares and solitaire. All are based on lovely animated dealing and the first computer card deck that really looks like a deck of cards.

The best of the adventure games comes out of left field, so to speak. Created in



England, The Pawn is being distributed in North America by Firebird. This is probably the first adventure on any machine to marry text and graphics in a really attractive manner. The graphic acts as an illustration... it stays out of the player's way until pulled down, curtain fashion, using the right mouse button. Otherwise the game acts like a pure text adventure, with a parser that is probably superior to the famous Infocom system. Graphics are loaded only when the picture display is in view, so you can trot through well known sections of the game with no annoying pauses for disk access.

The Pawn graphics are as well drawn as one would have a right to expect on the ST. The game itself is written in the classic Zork sword and sorcery format. The puzzles look rational enough, and the terrain certainly encourages exploration.

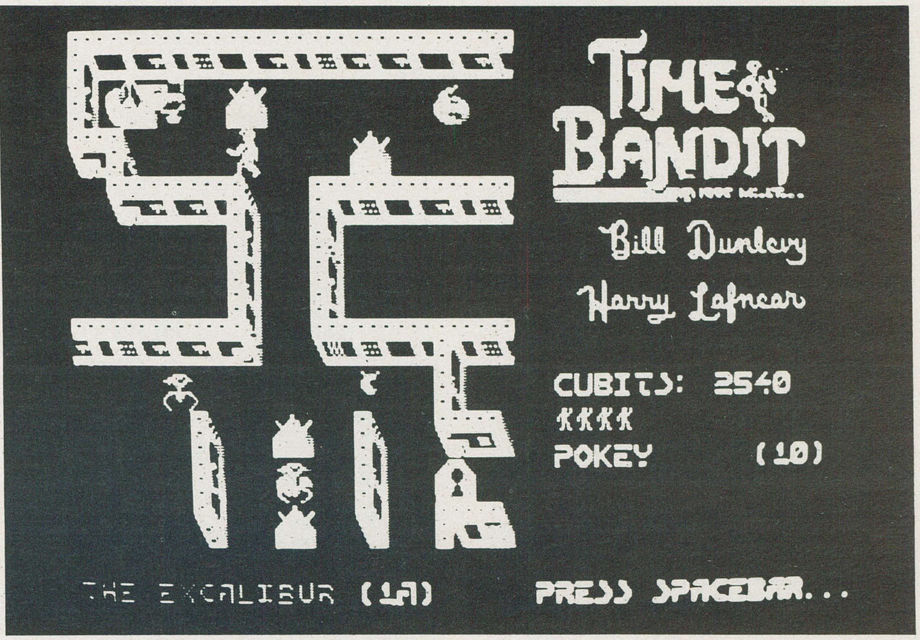
With all games, the price should come strongly into the reckoning. Games like Transylvania or Kadash would be worth having... just for laughs... provided the price was low enough. If you're going to pay the big bucks, I think you might want to hold out for real native grown Atari ST software.

Serious Stuff

Two standard business applications that seem nailed down quite well on the ST are spreadsheets and databases. The top two MS-DOS programs in each of these categories have been cloned for the ST, with apparently satisfactory results.

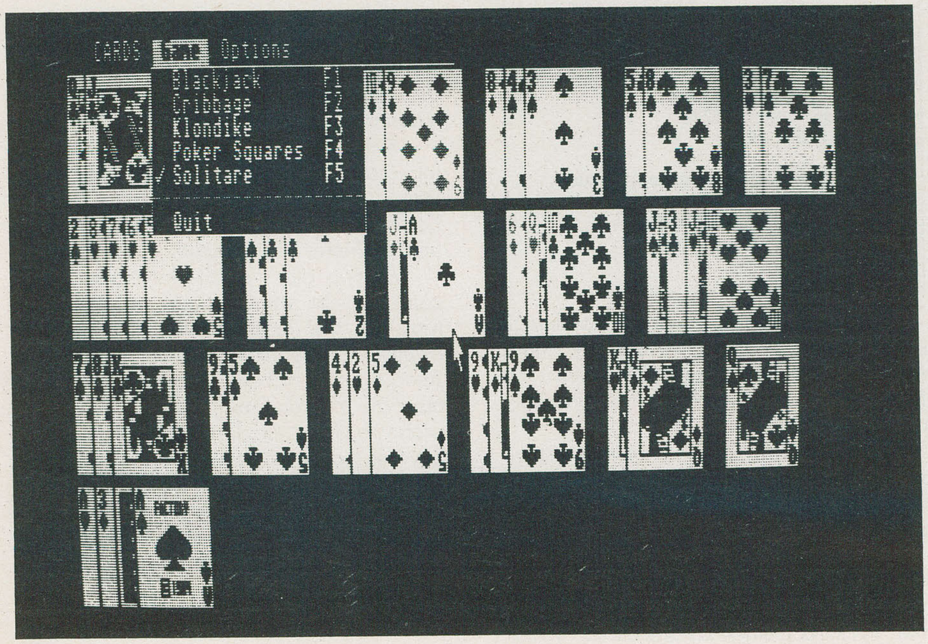
I haven't seen it... yet... but VIP Professional, from VIP Technologies, is said to be a pretty complete work on the lines of Lotus 1-2-3. The next release will go a step further, adding support for the ST's own desktop functions such as windowing and drop down menus.

Holmes and Duckworth has the other big business application more or less in hand, with its H&D Base... a clone of dBASE III. Written in H&D Forth, H&D Base includes virtually every dBASE II function you could name. Although it does not match up to the more recent dBASE III or III Plus standard, H&D Base has a unique advantage of its own. The H&D Base environment allows access to its Forth underpinnings through the *set forth on* command. Thus, in addition to the dBASE style programming capability, you can also use Forth operations. H&D Base is a massive system... we'll likely have a deeper peek into its innards in a future issue.



Word processing is in fair shape, considering that the 1st Word program included with the ST is not such a bad way to go. Atari has its own ST Writer... a straight up port of the earlier eight bit system. Regent Software has both a word processor and spell checker, but I've yet to see these and therefore can't offer much opinion as to their quality.

The most fundamental ST utility package so far is probably MichTron's Tool Box volume one, a compendium of five simple programs. You get file undelete... indispensable for those of us who get a bit too rash with the trash icon. Then there's a pair of byte editors, one for files and sectors and one for RAM. Both give a hex and ASCII display much like IBM's debugger. There's a format and copy program that integrates two TOS functions into one operation. Finally, there's a doohickey that will spit a disk directory listing into a text file, suitable for printing... one of those things one takes for granted in an operating system equipped with I/O redirection, but hard to accomplish on an ST desktop.



MichTron also has a selection of other utility products. These include M-Disk, a RAMdisk program, Soft Spool, a spooler, Calendar, a desktop appointment calendar and Mi-Dupe, a fast disk duplicator. The American prices for each of these are in the thirty to forty dollar range. The Tool Box is about sixty dollars.

Although we've not yet had a deluge of ST language systems for review, I can say that I've seen a few go by. The dealers will be able to supply you with just about any obscure sort of dialect you'd like. There are several good assembler systems on the market, improved BASIC, Pascal, numerous C compilers, a Modula or two... and of course, the ubiquitous Forth.

Technical languages such as Fortran... well, they might be out there somewhere.

Of the C systems, Megamax seems to be getting the most attention. It's only just become available, but got raves even in its beta versions. This is a port of the compiler also available on the Macintosh, and is the system used to create the game Megaroids on both the ST and the Mac. TDI Modula-2 seems to be the leading contender of the alternative languages. At least one of MichTron's commercial products is done in TDI Modula.

At least two commercial command line shells are out there as well, an MS-DOS emulator from MichTron and a Unix type system from Beckemeyer Development Tools.

The Atari ST Developer's Kit

The ST developer's kit is a key piece of software, but one that most ST users will never see directly. Atari has deliberately chosen not to promote the package too widely, partly because it hasn't had time to smooth out all the rough edges. The kit has the grease smeared look of an auto mechanic's toolchest.

The kit was undoubtedly a pretty good value in its time. In fact, it was certainly instrumental in ensuring a fertile software environment at the time of the ST's introduction. Although its importance will gradually decline as more third party developers take up the slack in both programming tools and documentation, it will continue to define the bottom line in system support. Would be developers would do well to check out this advance look at what they're getting into.

The developer's kit consists of five hand labelled disks. There's no attempt at slick packaging, but each disk is crammed full of the sort of nitty gritty tools that working programmers just can't get by without.

The first two disks contain the standard Digital Research C compiler and linker. There's a mess of standard header files to help deal with the complexities of the GEM environment. Separate link files are provided to allow creation of both stand alone and desk accessory programs. The compiler disk also includes a macro assembler, supplied by a company called Alcyon. There's a long assembler equates file that provides ready access to the important ST constants.

Both the compiler and linker disks include a batch processing program by Activenture that lets the developer automate the compile and link process. The batch program has the unique extension .TTP... for "TOS takes parameter". ST program files with this extension automatically pop up a dialog box to allow the user to enter all the stuff that would have been command line parameters in a command line system. Thus to do a one step compile, you'd double click the batch program, then enter the name of the compiler batch file followed by the name of the source file.

The batch processor needs two accessory programs. Since the desktop file delete function is not directly accessible from within batch programs, a tiny utility called RM is invoked to clear away all the intermediate object files after a successful compile. Another tiny program... WAIT... pauses and waits for the user to hit return.

A separate disk contains a choice of two different shape and icon editing systems. These should speed up the creation of all those graphic desktop doodads that GEM can't live without.

Sources:

Atari (Canada) Corp.,
90 Gough Road, Units 1 & 2,
Markham, Ontario L3R 5V5
(416) 479-1266

Penguin,
830 4th Avenue,
P.O. Box 311,
Geneva, Illinois 60134
(312) 232-1984

MichTron,
576 South Telegraph,
Pontiac, Michigan 48053
(313) 334-5700

Firebird Licensees,
P.O. Box 49,
Ramsey, New Jersey 07446
(201) 934-7373

Holmes & Duckworth,
Mirage Concepts,
4055 West Shaw 108,
Fresno, California 93711
(209) 227-8369

Megamax Inc.,
Box 851521,
Richardson, Texas 75085
(214) 987-4931

TDI Software Inc.,
10410 Markison Road,
Dallas, Texas 75238
(214) 340-4942

Beckemeyer Development Tools,
592 Jean Street 304,
Oakland, California 94610
(415) 658-5318

Yet another disk contains the default screen editor... MicroEMACS, by Mark Williams. This isn't exactly WordStar, but those with a Unix ancestry will probably love it.

A utilities disk contains numerous small but indispensable accessories. For instance, there's a rather funky command line shell program that emulates some of the functions of MS-DOS. To give you some idea of the state of refinement of the entire kit, I'll point out that this command line program fails to clear the screen when invoked. It just writes all over the desktop backdrop until the dead image scrolls mercifully off the screen. However, the command shell is still adequate to allow programmers to escape the sugar coated desktop environment.

Also on the utilities disk are a hex file dumper and a program that will let you search your files for a specified text string. Another two trivial programs switch the ST between high and low resolution display... a function normally accomplished from a desktop menu and hence unavailable for either batch file or command line execution.

The utilities disk also houses the Digital Research symbolic interactive debugger... SID for short... and a Kermit file transfer program.

Documentation for the developer's kit arrives as a foot thick stack of paper, xeroxed on both sides and three hole punched. This is the sort of quality printing job where crumpled original sheets are reproduced verbatim, looking like an illustration in a text on Origami. However, the information is all here... in frightening detail. You'll need about four big binders to stash it all in.

First of all, there's the standard GEM documentation. Volume one deals with the VDI virtual device interface. Volume two covers the more fundamental the AES application environment services level. Then there's the complete GEM DOS spec, marked "DRI confidential, internal use only." I shall prudently avoid any further description of the contents thereof. Then there's the famed "Hitchhiker's Guide to the BIOS", an overview of BIOS service routines and functions. After that we have the "Line-A Document", listing the fifteen primitive graphic services available via the 68000 line A exception handler. Finally, for the real keener, there's a complete source listing of the BIOS.

If hardware is more your thing you'll want to check out the "Engineering Hardware Specification". This covers the internal structure, ports and such, and ends off with a complete set of schematics.

Naturally, there are separate manuals for Digital Research C and CP/M 68K. Last... and probably least... there's a thin Kermit user's guide, originated by the Columbia Center for Computing Activities.

As you can see, the kit makes up in completeness what it lacks in elegance. Most of the higher level data is now available in friendlier formats... notably the Abacus series of books covering GEM and other ST innards. The Digital C compiler is probably being supplanted by Megamax C. Several top flight assembler systems have also turned up. Still, the kit does seem to provide everything a developer needs to go into business cranking out ST software. If you're really serious, the price is pretty reasonable, too.

CN!

A Great Buy With Strings Attached!

Start your Spring with a fresh approach to your music with a set of the new Labella Hard Rockin' Steel electric guitar strings and Canada's magazine for musicians – Canadian Musician.

Read about your favourite flourishing Canadian bands and artists. Open your mind to the latest tips from the pros on percussion, guitar, bass, keyboards, arranging, brass, woodwinds, recording, computers, vocal, sound & lighting and business. And that's not all!

Read about recording & equipment innovations; computer music, product reports, record profiles, industry updates and much more. You have the pick of the bunch.

Grow with us and strum along with your new Hard Rockin' Steel strings!

Watch out for our next issue featuring a series of articles on Computers & Music.

COUNT ME IN!!

Name ______________________

Address ______________________

____

City ______________________

Prov./State ____________ Code ____________

(Please check one)
Inside Canada
☐ 1 yr. $12.00 ☐ 2 yrs. $21.00 ☐ 3 yrs. $29.00 ☐ 5 yrs. $39.00

Outside Canada
☐ 1 yr. $15.00 ☐ 2 yrs. $26.00 ☐ 3 yrs. $36.00 ☐ 5 yrs. $49.00

Enclosed is my cheque or M.O. for ______________________

Charge to my MasterCard ☐ or Visa ☐ (please check one)

Card No. ______________________

Expiry Date ______________________

Signature ______________________

Complete and mail today to: CANADIAN MUSICIAN 832 Mount Pleasant Rd., Toronto, Ontario M4P 2L3

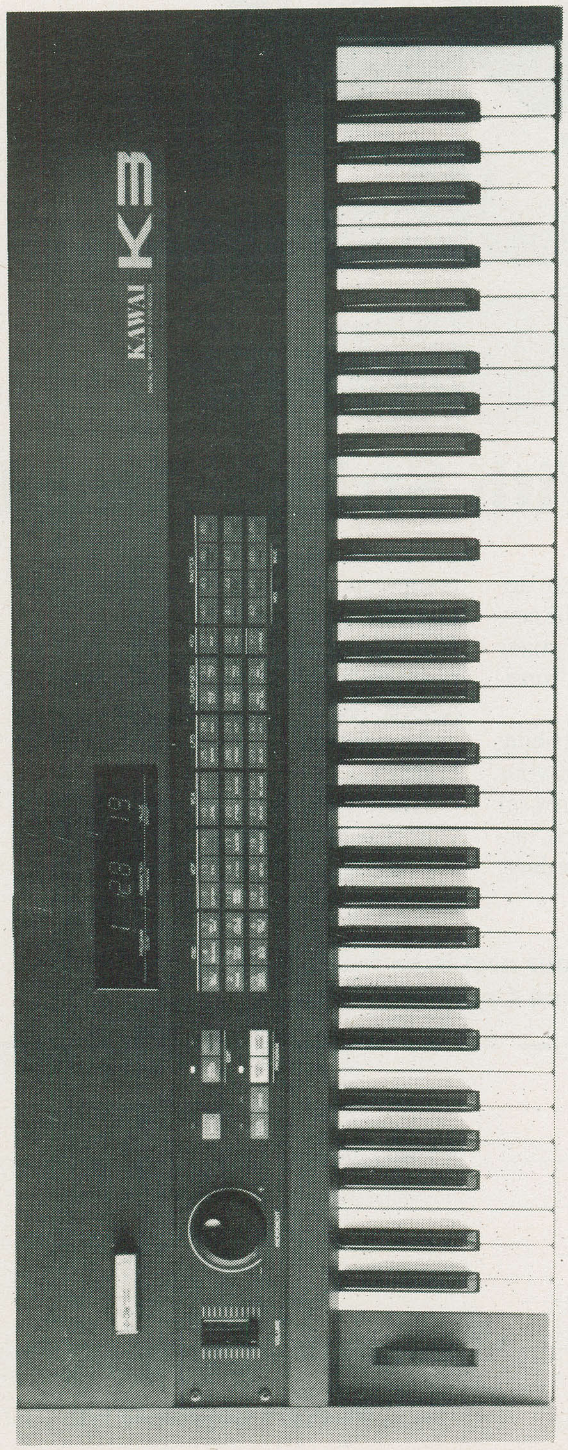
All MasterCard and VISA orders, call today (416) 485-8284.

Hard Rockin' Steel Strings Gage Guide: (Select One)
☐ Ultra Light – .008
☐ Extra Light – .009
☐ Regular – .010
☐ Medium – .010 (New! Blue Steel Wound)

The Kawai K3 Review

In the galactic quest for the ultimate electric piano, the K3 may look like just another legless plastic harpsichord. However, there's life in there, if of an unusual species.

by Steve Rimmer



The Kawai K3 is a heavy little mother. You'll probably notice that first. It comes in a massive double walled cardboard box that barely manages to contain it and it's built like the sorts of cars that people were scrapping ten years ago and presently pay a lot of money to restore. If you're thinking of taking one out to gigs, you'll either want a roadie or an athletic bass player.

Not surprisingly, the K3 looks like every other Japanese MIDI keyboard in creation. It has the obligatory wheel on the left side, a slot to plug in a memory cartridge and a row of membrane switches exactly where you'd expect to find them. There are all the usual connectors out back. The channel number display is orange, a unique variation.

Despite its visual similarity to everything else on the planet just now, the K3 is an interesting instrument. However, its place in the grand scheme of the universe is a bit unusual, and its powers lie in areas that some players may not know existed.

The Grand Illusion

The K3 is one of the least involved MIDI keyboards to play. You can get it together in seconds. Plug it in, turn it on, stab one of the fifty voice selection switches sprawled across the top of the case and boogie. Zap a MIDI cable into its back panel MIDI interface and your computer can make it boogie. If you can play a piano, you can have the K3 sound like a munchkin on acid with no further instructions.

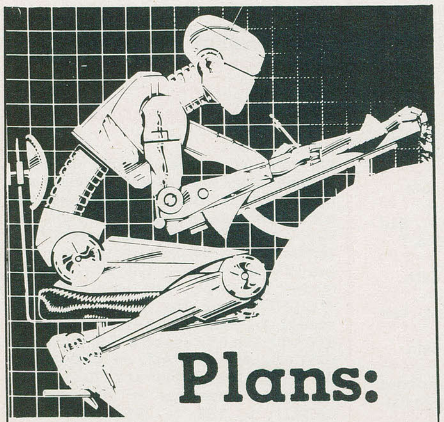
The K3 comes with fifty voices in its internal memory. They aren't the slickest voices in creation... few of the synthesizers I've tried came with good voices, though. Quite a number of them seem to have been chosen to sound a bit like the voices one finds in a DX-7. They do to some extent... the synthesis technique used by the K3 is similar to the one that the DX-7 is based on.

I wasn't wholly impressed with the ultimate ability of the K3 to realistically sound like things. You can improve on what's in the machine quite a bit... we'll get to that in a second... but the quality of the voices often winds up sounding synthetic. It's fairly good at predictable instruments... it does a pretty reasonable cello, for example, and it's quite decent on ensembles of

brass or woodwinds. It lacks the subtlety to do a really superb piano voice, however, or a human sounding human singer. Its harpsichord sounds worse than an old upright piano with tacks in its hammers, although in playing with it I ultimately came up with a more pleasing one.

The K3 I got came with a memory cartridge stuffed with another fifty voices. These were a bit more innovative than the built in lot. A lot of them still came off sounding a bit mechanical, but at least there were some authentically weird ones in there. The names the voice list gives them are good too... we have "flootz", "chillo", "klahvenet" and "ham n cheese", the latter being a sort of Hammond B3 voice.

Some of these voices are authentically useless, but, of course, the real glory of editable voice parameters is that you can zap the ones you don't like and replace them with better noises. Quite a few of the voices in the memory cartridge were more special effects than playable sounds. At least, I wouldn't have wanted to have tried to play them.



Hardware:	**K3 Digital Wave Memory Synthesizer**
Manufacturer:	**Kawai Canada Music, 6400 Shawson Drive, Unit #1, Mississauga, Ont. L5T 1L8 (416) 673-2345**
Price:	**$1,795.00**

The real power of the K3 is in its voice editing. Whereas the Yamaha instruments have marginally editable parameters... if you really want to... the K3's voices can be manipulated with relative ease. It would still be easier with a computer program to handle it for you, but the task is of a manageable hugeness using only the membrane switches.

Whereas the DX-7 wants one to express its voice parameters as fundamentally weird FM synthesis parameters, the K3 lets you deal with things in much more familiar synthesizer terminology. There are virtual oscillators to manipulate, and you can determine the harmonic content of voices with the digital equivalent of filters, rather than with DX style "operators".

The K3 does a good job of implementing all of the MIDI parameters as programmable voice characteristics. You can determine things like touch sensitivity. There are also a number of useful real world effects in there, including a chorus function, a delay line and so on.

Editing voices on the K3 is kind of a blast. One could wish for sound synthesis circuitry that was capable of slightly better things, but in some cases the ease with which one can manipulate it makes up for some of its shortcomings. It takes a while to get used to the techniques of relating the parameters the K3 offers one to actual sounds, but once you've got that together you can concoct voices pretty quickly.

The Key

The K3 is a generally good instrument to play... like many of the latest generation of MIDI keyboards, it's a bit better suited to pop than to anything else. Its facility of having lots of voices on line in memory cartridges is handy. It allows one to edit the internal voices and subsequently restore the original fifty factory presets if one gorches the noises beyond redemption... a handy feature. The voices lack the facility of storing names with them... that's one of the things I would have lifted from the DX instruments.

The MIDI implementation on the K3 appears to work pretty well. As with the DX-7, the voice parameters can be dumped over the MIDI bus, allowing them to be manipulated by an external computer. However, to the best of my knowledge a K3 voice editor package doesn't exist. The K3 is considerably better at handling voice changes while it's playing than are many synthesizers... even really unfair pieces, things that change voices in between sixteenth notes, don't seem to upset it unduly. It allows one to select a mixture of its internal and cartridge voices on line. The cartridge voices have numbers starting at fifty-one.

At almost two grand the K3 isn't cheap... words being such conceptual approximations, I'd recommend having an extensive listen to one before you go applying for a credit limit extension on its behalf. However, the K3 is a decent machine, and well worth considering if you have a large flat space that wants filling with a MIDI synthesizer. **CN!**

Write To Us

Aside from the usual blue pencil and classic Coke, one of the most important things to a magazine editor is feedback. It's extremely hard to know what is being well received in a magazine and what is not.

You can help us make Computing Now! better.

Before you forget, take a few minutes to write and tell us what you think about this magazine... the good stutt and the nasty bits. We'd like to hear your suggestions and complaints, ideas for future articles, things you miss or would like to miss.

Send your comments to:

The Editor
Computing Now! Magazine
1300 Don Mills Road
Toronto, Ontario
M3B 3M8

COMPUTER
PARTS GALORE
316 College St. INC.
Toronto, Ontario M5T 1S3

Toll Free Orders Only
1-800-387-1385
(416) 928-2161

KEYBOARDS

- (A) **5150 Style,** 83 Keys, Std Layout, uses the long lasting Cherry Sw, IBM® Comp. .... $ 99.95
- (B) **5151 Style,** 105 Keys, Keytronic 5151 layout. Cherry Sw. IBM® Compatible $139.95
- (C) **5160 Style,** AT® compatible, with 83 keys by Cherry, 5150 layout $159.95
- (D) **5161 Style,** AT com, but 5151 layout $189.95
- (E) **AES DATA** Keyboard, Par ASCII and Serial but not IBM. Runs Apples® very good general use ASCII keyboard .... $ 24.95
- (F) **APPLE NUMERIC BASIC FUNCTION, II+** .... $ 69.95
- (G) **APPLE STANDARBASIC FUNCTION, II+** .... $ 69.95
- (H) **APPLE DETACHED II+, NUMERIC FUNC.** $149.95
- (I) **FOX 2001** Internal keyboard, IIe .... $ 69.95
- (J) **FOX 2001** External keyboard, IIe $139.95

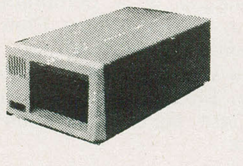
POWER SUPPLIES

- (A) **150 Watt, Back/Side Switch,** with +5v-15Amp, +12V-5.5Amp, -5,12V-½Amp, A good quality, hydro approved unit $139.95
- (B) **150 Watt, CSA** approved unit, for the very highest system reliability, +5v-15Amp +12v-5.5Amp, -5,12V-½Amp, Side Sw .. $159.95
- (C) **APPLE II+, or FOX 2001 IIe,** power supply, +5v-5Amp, +12v-2.5Amp, -5, 12V-½Amp Very high quality .... $ 79.95
- (D) **OPEN FRAME** Northern Telecom switcher as widely sold, +5v-4A, ± 12V-1Amp and -5 at 1 Amp, very good deal .... $ 18.95
  Power cord for (D) above $ 3.50
- (E) **IBM OPEN FRAME,** came from the defunct NCR compatible, +5v-7Amp, +12v-3Amp, -5, 12v-½Amp, runs the whole thing .... $ 59.95

V-20 CHIP

You may have heard of the NEC V-20 chip. It is an upgraded 8088 that can run programs up to 50% faster (depending on code). It also can run Z-80 code, allowing it to be used for Z-80 development. Especially good for Video when using code using it's features. A hot and hard to get CHIP at (5 Mhz) $24.95
Faster version runs at 8 Mhz .. $44.95

HARD DRIVE BOX



This nicely made box is ideal for externally mounted hard or floppy drives. It has space for one full height or two half height drives and power supply space at rear. This is a well made and rugged units for only $59.95.

CASES

XT® CASE WITH AT® LOOK

- (A) Standard 8 slot hinged case XT® case. Fits side or back power supply (Specify) Has much heavier metal than competitors for stability, takes 1-4 ½ sized drives and fits very well $79.95
- (B) Same as above, but the front looks like an AT case, with switch for KB and LEDS on front (picture) $89.95
- (C) ABS numeric case for II+ $49.95
- (D) ABS case for FOX 2001 ... $49.95
- (E) Hard drive box, holds 1 full sized or two ½ sized hard drives, good for portable tape backups as well $59.95
  40 Watt power supply for the above $59.95
  60 Watt power supply for the above $79.95

DRIVES QUME

- (A) **Yes** we have the drives that IBM bought for their own use, complete with the IBM® logo right on them. Brand new DS/DD ½ Height drives $129.00
- (B) **Panasonic** drives, very quiet, as above but no logo, DS/DD ½ Height $165.00
- (C) **Toshiba** drives, noiser than above, but cheaper, no logo, DS/DD ½ Height $159.00
- (D) **Seagate** 20 Meg hard drive with Controller, ½ height $699.00
- (E) **Seagate** 20 meg drum only $549.00
- (F) **Controller** only $169.00

GRAY SCALE ADAPTER

As you know the IBM colour graphics card does not look very good on a monochrome monitor. This is because often two colors look the same on monochrome so you cannot read red print on a blue background or some such. This little gray scale proportinately scales each color into a different intesity allowing easy viewing on amber, green or white monitors. We sell it as a kit for only (with cable) $19.95
or wired and tested (with cable) $24.95

RIBBONS

Gemini 10X	$ 3.95
Gemini 15X	$ 3.95
Centronics 739	$ 6.95
Centronics 150	$ 7.95
LX80	$11.95
Olympia	$ 8.95
MX-80	$ 8.95
TTX	$ 7.95
Diabolo	$ 6.95
CITOH Imag Wr	$ 6.95
QUME IV	$ 8.95
Tally 1000	$ 7.95

MODEMS

GVC INTERNAL 300/1200	$325.00
GVC EXTERNAL 300/1200	$325.00
SMARTMODEM 300	$249.00
SMARTMODEM 1200	$649.00
SMARTMODEM 2400	$995.00

CABLES

Disc Drive cable (3 Con & twist)	$12.95
IBM Parallel printer cable	$14.95
IBM serial printer cable	$14.95
IBM 6' Keyboard extension (curly)	$ 9.95

® Note: The terms, IBM, XT, AT, PC are registered trademarks of International Business Inc.

WIRED CARDS FOR IBM®

- (A) **384K RAM card, 3/4 size for 384K of 64K DRAM. Wired and tested with OK** .... $ 79.95
  Bare PCB and assembly data .... $ 19.95
- (B) **576K RAM** card, ¼ size for 576K of 256K DRAM. Wired and tested with OK .... $ 59.95
  Bare PCB and assembly data .... $ 14.95
- (C) **KRAM card, a clone of the JRAM card for 2 MBYTE of 256K DRAM. Runs all software 100% compatible, W & T with OK** $139.95
  Bare PCB and assembly data .... $ 29.95
- (D) **COLOR GRAPHICS,** A PERSYST clone, runs all software and our one has grey scale O/P & color comp and TTL RGB, W & T $125.00
- (E) **MONO GRAPHICS,** A hercules clone, runs all software and has parallel centronics printer as well, W & T $139.95
  Bare PCB and assembly data .... $ 24.95
- (F) **384K/MULTIFUNCTION** card, exact copy of AST SIXPACK, Ser, Par, Clock, Cal, Game I/O with software and ports, W & T, OK. $169.95
  Bare PCB and assembly data .... $ 24.95
- (G) **DISC CONTROLLER,** runs 1-4 drives has newest 9216 data separator with cables wired and tested .... $ 59.95
  Bare PCB and assembly data .... $ 12.95
- (H) **DISC + I/O Card,** A very nice new card with disc controller plus AST sixpack on board without any RAM. Ideal for the new 640K motherboards, Ser, Par, Clock, Cal, Game, software and ports, W & T $189.95
- (I) **10+2,** An AST SIXPACK with no RAM, has Ser, Par, Clock, Cal, Game and software and ports, W & T $129.95
  Bare PCB and assembly data .... $ 19.95
- (J) **PRINTER,** Centronics, Std parallel printer, ¼ size, W & T .... $ 49.95
  Bare PCB and assembly data .... $ 12.95
- (K) **SERIAL,** std serial card, two ports ½ sized, W & T .... $ 44.95
  Bare PCB and assembly data .... $ 12.95
- (L) **CLOCK/Calendar,** A ¼ sized card for these two functions, with battery .... $ 49.95
  Bare PCB and assembly data .... $ 12.95
- (M) **GAME,** card, A ¼ sized card with 2 std game ports on it, W & T .... $ 39.95
  Bare PCB and assembly data .... $ 12.95
- (N) **EXTENDER,** and 6" extender board for easy access for service .... $ 14.95
- (O) **PROTOTYPE,** full sized with all WW area and a DB25, DB9 footprint .... $ 24.95
- (P) **RAM/PROTOTYPE,** Full size with 256K/1M RAM area for 64K/256K DRAM at end with DB25 and DB9 footprint .... $ 29.95

CENTRONICS CONNECTORS

36 Pin solder cable male	$4.95
36 Pin Flat cable male IDC	$4.95
36 Pin Flat cable female	$4.95
S-100 Edge Connectors	$ .99

PRINTERS

ROLAND

PR1212A	$575.00
PR1011	$375.00
PR2022	$645.00

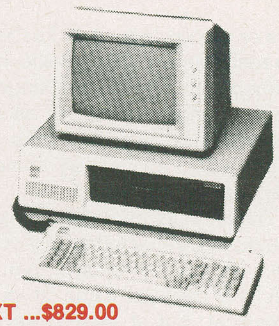
OLYMPIA

NP	$425.00
RO	$475.00

CITIZEN

MSP 10	$450.00

IBM COMPATIBLE SYSTEMS



XT ...$829.00

What a deal, the nicest systems of all at the lowest price for good merchandise. The basic system has the following. For **$829.00**

- **640K Motherboard**
- **1 DS/DD 360K ½ Ht.DD**
- **1-4 DD Controller**
- **Keyboard**
- **Colour Graphics Card**
- **Free software**
- **256K RAM on board**
- **8 XT slots**
- **Flip top case**
- **150 Watt power supply**
- **All cabling & 24 hr. Test and burn-in**

To upgrade the system to your choice please select from these options.

- **Monchrome graphics card** . Add $ 49.00
- **Ser/Par/game/clock/calendar** .... Add $110.00
- **Printer card** .... Add $ 45.00
- **Serial card** .... Add $ 49.00
- **Second drive** .... Add $150.00
- **KRAM CARD, OK** .... Add $139.00

640K XT MOTHERBOARDS

We now have several types of 640K motherboards with all the standard attributes of the "REAL" thing as well as some new ones. All have 8 XT slots, XT size, and 8087 socket and have 100% compatibility. The extra featurs are the use of 256K RAM to get 640K on the mother board plus the new desirable "TURBO" feature. The "TURBO" has two speeds, the basic 4.77 and either 6.67 or 8.00 Mhz. The 6.77 uses the -3 or standard 8000 chip set and RAM, the 8.00 needs the -2 high speed version and 120Ns RAM. The speed advantage is the ratio of 6.67/4.77 = 1.39 or 39% faster and 8.00/4.77 = 1.67 or 67% faster, all other things being equal. Of course timing loops will be changed so that any real time output will be faster by the ratio unless the loop is fixed. The largest saving is in CAD programs with a lot of number crunching, smallest savings are disc intensive programs.

SME-XT, wired with OK 3 hr burn-in $199.00
SME-XT, wired with no IC's at all . $ 99.00
SME-XT, bare pcb, book parts list . $ 34.95

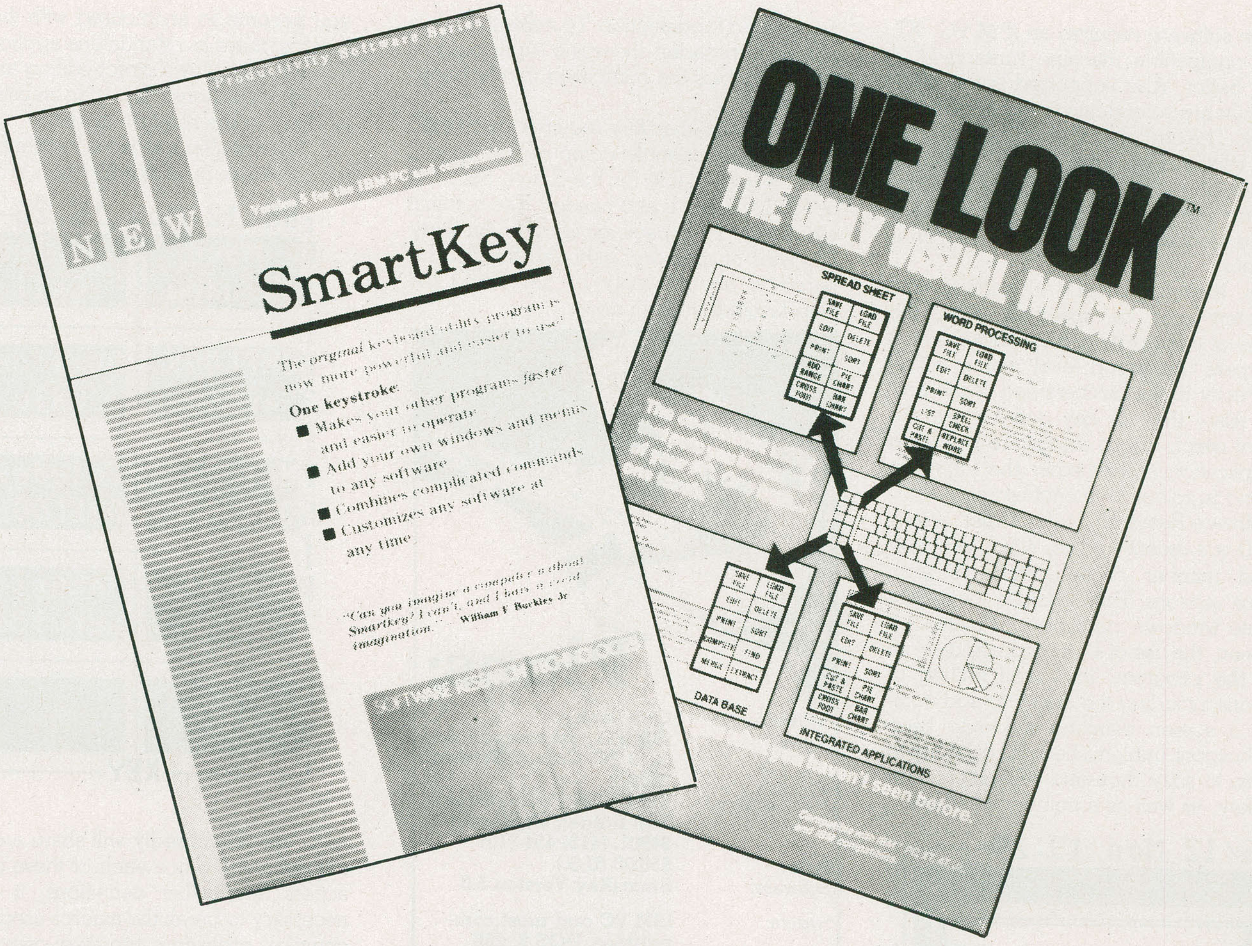
TURBO 6.67 MHZ VERSION
SME-XT A Wired, 3 Hr burn in, OK . $229.00
SME-XT A Wired, no IC's at all $129.00
SME-XT A Bare PCB, book, parts list .... $ 39.00

TURBO 8.00 MHZ VERSION
SME-XT B, wired, 3 hr burn in, OK . $249.00
SME-XT B, wired, no IC's at all ... $139.00

All boards come with book and parts list, the wired with OK have BIOS included.

APPLE WIRED CARDS

Applecard	$149.00	16K	$ 39.95
80 Column	$ 59.00	128K(OK)	$ 55.00
80 Column with softswitch	$ 69.00	128K(64)	$ 99.00
Grappler	$ 49.00	128K(128K)	$149.00
Cable above	$ 19.95	Drive Cont.	$ 49.95
		Z-80	$ 45.00



Two Macro Managers

Keyboard macros on the PC are among the most powerful tools in creation... especially if you don't type as quickly as you might like to. Here's a look at two of them.

by Donald Roy

Macro managers offer the possibility of saving many keystrokes while operating an application program. By assigning often repeated key sequences to a special key, the manager will enter this sequence for you. More than this, however, a capable macro manager provides an overlay environment that, in essence, can make another package programmable... within limits.

Here, we look at two programs that you can bolt onto your current software to both save repetitious keying and increase their degree of intelligence. *One Look*, from IMSI Software Publishers, takes a visual approach to handling macros and offers some handy features. *SmartKey 5.0*, from Software Research Technologies, provides a broad power base and the ability to customize software to your liking.

Appearances

One Look, from the very beginning, offers more than a simple macro manager. Aside from the needed internals to define, store and execute macros, the package also offers the services of a RAM resident notepad, calculator, telephone index and security services. As well, if you currently are using RoseSoft's *ProKey* program, these files can be converted for *One Look's* use.

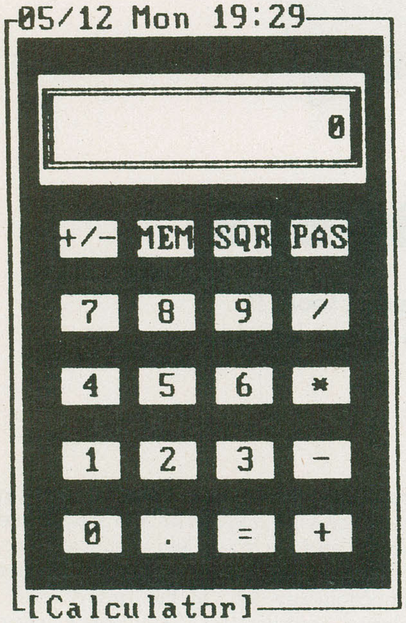
A point of difference between *One Look* and other macro oriented software is the visual approach taken in the user interface. This is achieved by using a graphic representation of the PC's ten function keys, to which titles and keystrokes are assigned. At the root is a group level menu which can call further menus for specific applications. Function key ten, at the group level, is always reserved for *One Look's* internal operations.

These internals include the RAM additions mentioned before, as well as the needed abilities to define and implement application templates. The window display can be moved into any of the four extreme corners of the screen, if the default upper left position is found distracting. In addition, the hot key or the key sequence used to wake the program from deep memory, can be adjusted to the alternate key sequence that pleases you.

The software requires an IBM PC, XT, AT or compatible machine running DOS two or better. Also supported is the 3270 PC, if you are so endowed, and the Corvus network. The cost of running the program is fifteen kilobytes of memory for the main program, plus additional space for the extra bits. Only the necessary parts of *One Look* need be loaded, if memory conservation is paramount.

On loading, several command line parameters may be specified. First, the filename of a template can be added which will bypass the group level menu, taking you directly to the business end of the program. Also, the hot key can be set, the speed at which macro strings are passed to the application and whether the tube you are using is colour or monochrome.

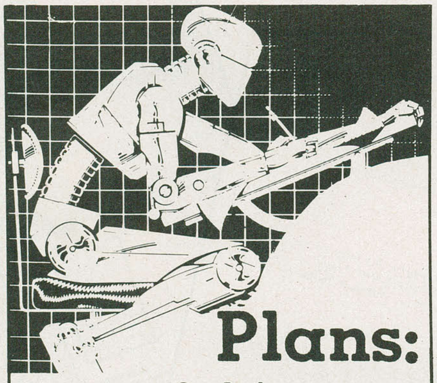
Software Research Technologies' *SmartKey 5.0* is a recent update, which provides a good number of enhancements from previous versions. These include the facility to create windows, prompts and menus on the screen, the use of DOS commands from within the program, improved editing of definitions including the ability to add comments, a screen blanking mode to keep your phosphor unblemished, the capacity to store up to sixty thousand characters to a single key, as well as a number of others.



Aside from the main program, the distribution disk also holds a *ProKey* to *SmartKey* conversion routine, a file encryption program, a keyboard reassignment module including a version for the Dvorak layout, a tutorial for those who hate reading manuals and several sample definition files.

If you have a superb memory, you will appreciate this package's flexibility in allowing almost any combination of keyboard characters, function keys and special keys... alternate, control, shift, home and so on... to be selected for definition. To make the selection even broader, there is a supershift key which doubles the number of combinations available.

Running *SmartKey* requires an IBM PC or fuzzily compatible computer. A special version is available for the Sanyo 550. PC or MS-DOS two is also needed. The nominal overhead for the package is just under thirty kilobytes of memory, although this can be pared to about twenty by sacrificing the help facilities and windows.



Plans:

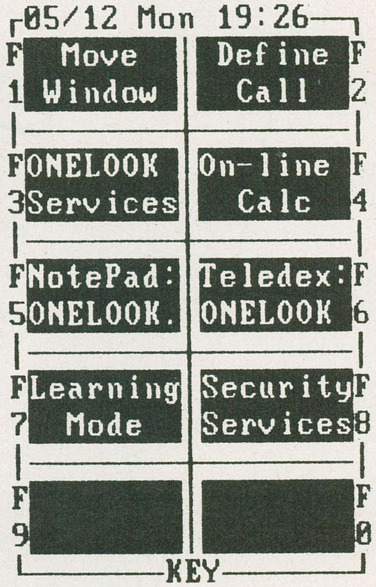
Software:	**One Look**
System:	**IBM PC and compatibles, 3270 PC, Corvus Network, DOS 2**
Manufacturer:	**IMSI, 1299 Fourth Street, San Rafael, California 94901, (415) 454-7101**
Price:	**$50.00 (U.S.)**
Software:	**SmartKey Version 5.0**
System:	**IBM PC and most compatibles, DOS 2, 28K memory to spare**
Manufacturer:	**Software Research Technologies, 3757 Wilshire Boulevard, Suite 211, Los Angeles, California 90010, (213) 384-5430**
Price:	**$49.95 (U.S.)**

If you're running older software you may appreciate this program's ability to add DOS function access to anything else you are using. Full directory access, including subdirectories, can be had for DOS versions two point whatever and above. Also available are the TYPE, CHDIR, ERASE and RENAME commands. A pointed variance from the DOS protocol is that *SmartKey* will not support the wildcard characters "*" and "?" when erasing files. This approach is quite intentional, preventing novice users from accidentally taking a Rambo approach to their disk files.

Operating Room

The process of defining macros and presenting them for use, with *One Look*, is simple enough for anyone with a clear understanding of what all the buttons on a PC keyboard are normally for. Tumbling through the internal selection from the *One Look* display, then choosing "template services", presents the functions needed to begin writing your own time savers. Should you become so enamoured with the use of macros, there is a standalone application on the disk that allows direct editing and entry of key definitions. This can substitute for working in the resident mode.

The first step needed is to define a name for some group of ten function key defini-



tions, that supposedly will share a common environment. Since each of these can only access ten further definitions, it may be necessary to break the macros used in more complex programs into their own groups. For example, all the macros used in Lotus graphing would be reasonable, where *all* Lotus macros would not. From here, individual definitions can be made for each of the ten function keys. Hitting control escape marks the end of the macro string input. Hitting the *Ins* key may be used to mark a sequence for interactive input from the user, such as typing a filename.

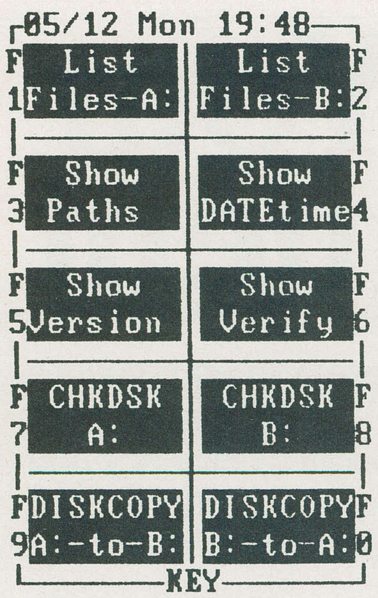
Control characters and function keys can be included in the definitions. As well, while the number pad keys are normally denied use, they can be included by pressing the scroll lock key. This, of course, is quite necessary in defining macros for spreadsheet use. Pressing escape signals the program that you have completed the input for the key of interest. At this point, *One Look* will signal its understanding and advise you of the current amount of RAM space available for other definitions.

Other functions available in the template services group allow you to move your macros from one function key to another, as well as loading, saving and clearing templates from memory. Standard issue templates are included with the program for such favourites as WordStar, DOS, dBASE II, Framework, Symphony and Multimate. Saved files are assigned a .KB file extension, which is not negotiable.

The other resident utilities included in

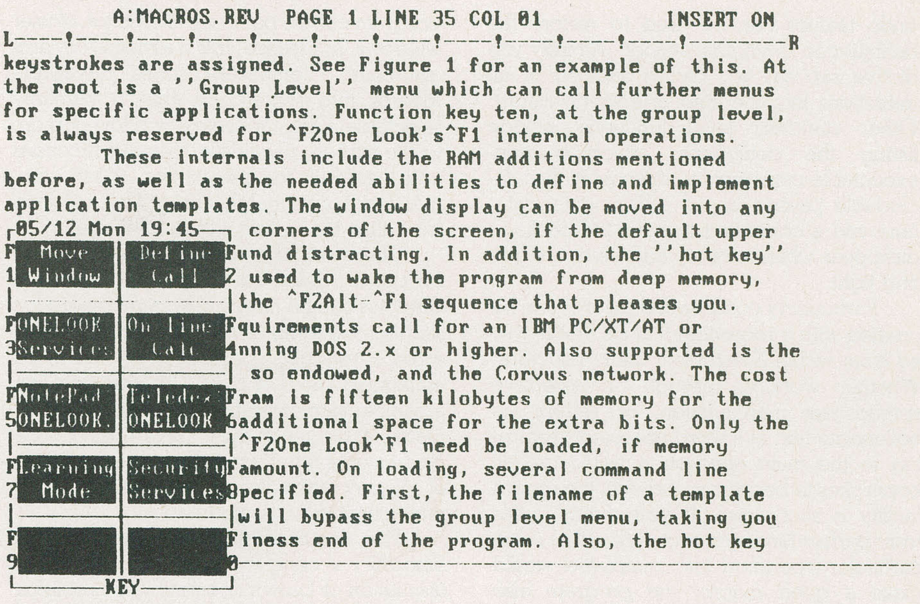
the package are useful, if not inspiring. The notepad will allow for the creation of files up to fifty lines long and includes the ability to transfer the content of notes into the current application program. Any notes created can be read from or saved to disk and sent to your printer for permanent glorification. The calculator offers standard arithmetic functions, square root, memory and pasting of the result into an application. Although pasting from the notepad will insert the desired text at the destination point, doing so with a calculated number will overwrite anything in its way.

One Look's resident telephone index program will support multiple files of names, addresses and phone numbers... though each is limited to sixty-four entries. With suitable formatting of the phone number and a Hayes compatible modem, the program will dial the number for your pleasure. A two second pause is available for accessing outside lines through a PBX. A find feature will search the phone list, given a first or last name, and display the first entry that matches. The security function that is built into the package will accept a password of up to eight characters. When enter is pressed following this, the screen will blank out, waiting for an escape to be



pressed. At this stage, the needed password has to be entered again in order to access the application that was running. This allows one to lock out other users from your unit while you are away for lunch. This protection can be disabled by rebooting the computer, however.

To ease the task of defining macro sequences, a separate module can be loaded with *One Look* that will establish keystroke recording, from within a spreadsheet or other program. With the appropriate procedures implemented, the routine can be called alternate equals sign. Alternate minus



WordStar with a macro menu.

sign will terminate recording. The resulting macro will be assigned to a free function key at the discretion of the program. If all ten keys happen to be defined for the group requested, the procedure will not be saved.

Overall, the approach taken by *One Look* makes a good degree of sense. With suitable forethought, macro libraries can be organized in a logical fashion that would not require extensive training or memory exercises if another person were going to use the macros you have defined. The program allows others to make use of the PC's function keys, if such support is not available otherwise. The strings assigned to a key may be up to fifteen hundred characters in length, which suggests their use in assembling boilerplate documents with a low end word processor. Throughout my use of the program, it did not exhibit any untoward behaviour, whether used on a real PC or a compatible.

Thinking Man's Keys

Often, setting arbitrary limits, which one honestly believes will never be exceeded, leads to trouble of some kind. Such a case is to be found in the memory limits of DOS, which Microsoft believed at the time was more than anybody would ever need, or want, for that matter. One of the ways... hopefully... to avoid this kind of problem, is to set such limits ridiculously high... and then increase them ten times. Perhaps this is how the limits of *SmartKey 5.0* were set.

If one were persistent enough, the program is supposed to allow macros of up to sixty thousand characters to be associated with a single key. There may be some real use for this ability, but at this time, it escapes me. However, it is unlikely that the company will ever be annoyed by a user complaining about needing one more character in his definition.

Similarly, the number of definable key combinations easily exceeds five hundred. In today's realm of complex software, this is not unreasonable, in order to find key combinations that are not used by the stable of your mainstay programs.

Defining macros under *SmartKey* is a straightforward process. When the program is called to attention, direct entry may begin, or upon pressing the numeric keypad plus key, you are presented with a Lotus style bar menu. From this point, one may elect to return to the foreground application, enter, list or edit macros, as well as selecting from several options that tailor the program to your particular needs.

Whether providing definitions from the initial display or using the edit mode, pressing the key, or key combination, to define begins the process. Subsequent keystrokes are recorded in the display window and the keypad plus key is again invoked, this time to terminate the entry. Key definitions are stored in memory until saved to file. Extended characters can be entered by pressing the alternate key and entering the ASCII number of the blob you want.

The keypad minus button is assigned a special status as the supershift key. This route allows the definition of another whole set of macros that overlap the existing ones. Thus, one could have callable macros for control A, alternate A and the supershifted versions of the same key combinations. Another use of this key is to make the program assume temporary amnesia while you employ some keystrokes in their original meaning.

Once defined, macros can be edited without retyping the entire definition. Given the sixty thousand character capacity that may be assigned to one key, this will be found a most handy inclusion. Should you get a bit lost somewhere along the way, an

undo feature can be used to restore the definition to its original version. A really vindictive zap call will banish selected or all definitions into the land of ancient history. Other standard level functions include listing the definitions, attaching non-executable comments to them as well as includable pauses, speed adjustments, date, time and a command to have the package disappear when memory addresses are getting tight.

Particularly complicated macros can be created with a recording feature that is called from within *SmartKey*'s menu structure. Whether working through an extensive spreadsheet manipulation, or setting up boilerplate text in a word processor, returning to the main application will cause all keystrokes to be memorized until the record facility is shut down. If you should suffer from perfect hindsight and realize that some particular operation just completed would make a good macro, the program also keeps tally of the last sixty-four keystrokes. These buffered contents can be transformed into a macro definition after the fact.

Already mentioned, *SmartKey* provides access to several DOS functions that prevent the necessity of exiting the current application to change directories, rename files and so on. Another advanced use the program can be put to is the creation of user windows. In essence, this is a method of programming a complicated series of available macros into some form that is humanly selectable. One can have a window pop onto the screen, provide a menu of functions and then wait for instructions. When commanded appropriately, *SmartKey* will then look after the grittier details of the job.

There are other uses that this facility can be put to. Since the disappearance of the windows can be timed, they can be used for simple messaging. A full set of resident help screens could be constructed by the more ambitious. The distribution diskette contains a sample demonstration file that highlights many of the techniques. Windows can be made to call other windows and full support of display attributes is provided in both monochrome and colour.

The documentation that accompanies *SmartKey* is complete in all regards. A full discussion of known compatibility problems is included and typically revolves around resident or foreground applications that bypass the DOS keyboard interrupt to handle input from the real world. The current version is incompatible with non-standard video cards, such as the Hercules sort. A full set of examples is provided within the paperback collection of pages to take you through most every aspect of working with the program. While a bit of persistence may be required in some areas and the visual presentation could be more understandable, there are many worse manuals around.

A number of other functions are provided with *SmartKey* that add value to the package. Overall, this is a powerful addition to most any software library.

CN!

Circle No. 8 on Reader Service Card

The Book Of Computer Music

MIDI may be the most significant advance in music technology since the discovery of sound. However, if you can't understand what it's capable of... or can't comprehend the cacaphony of manufacturers' claims that surround it... it might as well have been a new revolution in food processors.

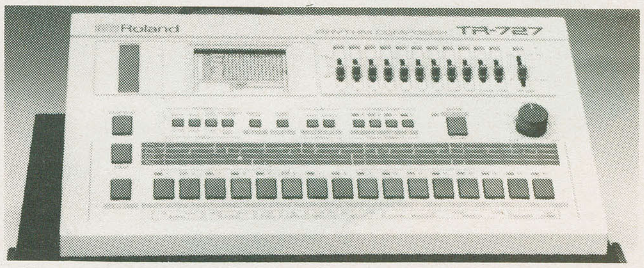
We understand MIDI better than anyone. We're comfortable with both the computers in all their confusing technological glory and with the music that makes them all worth while. If you read the **Book of Computer Music** you'll understand MIDI too.

And you'll realize that it's as magnificent a trip as everyone says it is.

Whether you're an amature musician with a cassette deck and a keyboard, a working performer or a studio artist, the **Book of Computer Music** will turn you onto MIDI as no other reference work can. By the time you hit the back cover you'll be wholly comfortable with system exclusives, tracks, MIDI messages, FM synthesis, through boxes and all the other mystifying technological mind games that accompany this powerful new musical instrument.

You'll be able to play MIDI as easily as you play keyboards for guitar... or flute or violin or percussion or nose harp... right now.

Included in the **Book of Computer Music** are articles about



The Yamaha DX-7 • The IVL Pitch Rider • Personal Composer • The Yamaha RX-21 • Total Music for the Macintosh • The Yamaha CX5M Music Computer • The Akai AX-60 • The Roland GR Instruments • Computers Fopr Musicians • The Roland SDE 2500 MIDI effects system • The Yamaha DX-100 • The Nexus FM Drawing Board • The Akai S612 Sampler • Building a MIDI Through Box • The Roland Alpha Juno • DX-7 Voice Editing • Writing Your Own MIDI Software.

Plus a complete introduction to MIDI in a number of in depth, easy to understand articles that won't leave you feeling like your head got lost in the Twilight Zone.

Give us five bucks... we'll make you smart

Actually, we only want

$4.95

plus 7% music tax for Ontario residents and $1.00 for postage.

The Book of Computer Music is available now!

To order your copy, please send $4.95 (plus $1.00 postage and 7% Ontario sales tax) to

The Book of Computer Music
1300 Don Mills Road,
Don Mills, Toronto, Ontario M3B 3M8
or phone us at (416) 445-5600

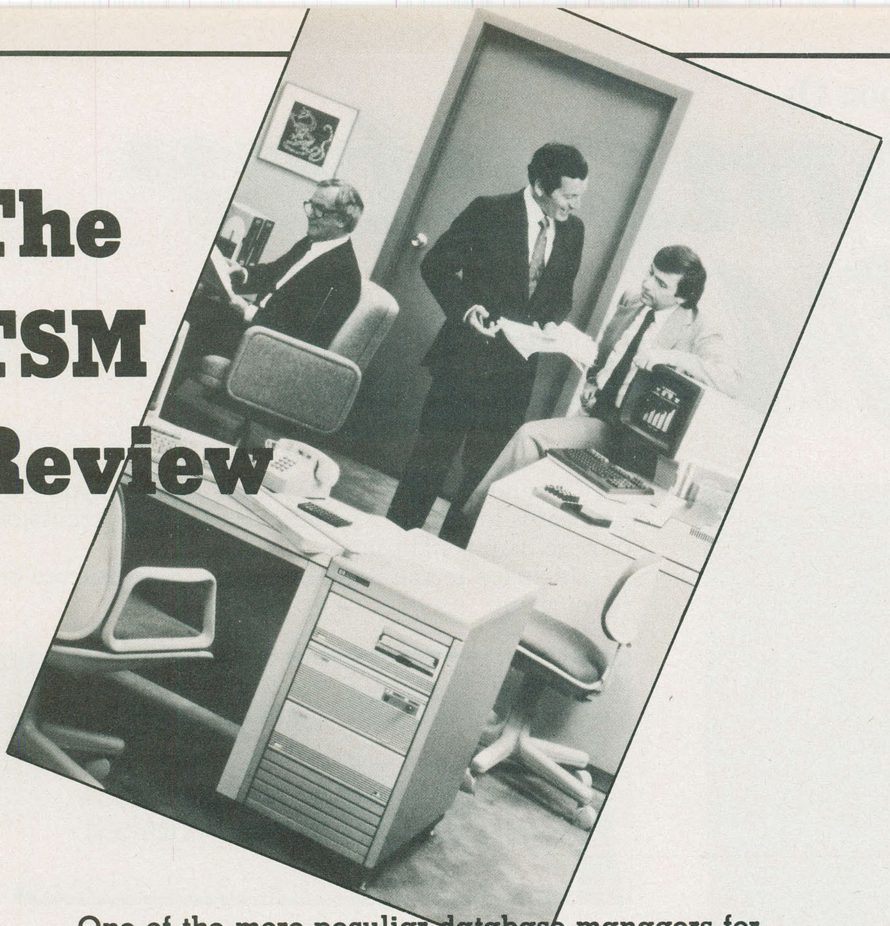
NAME ______
ADDRESS ______
TOWN/CITY ______ PROVINCE/STATE ______
POSTAL CODE ______ DATE ______

☐ Cheque enclosed DO NOT send cash
☐ Mastercard Account No. ______
☐ Visa Account No. ______
☐ American Express Account No. ______
Expiry Date ______
Signature ______

Moorshead Publications, Subscription Department,
1300 Don Mills Road, Toronto, Ontario M3B 3M8 (416) 445-5600

MH/07/08

The TSM Review



One of the more peculiar database managers for the IBM PC, this one has an odd mix of features.

by Frank Lenk

TSM, from DynaBase Superior Software Systems, is a very peculiar package... one that defies any simple yea or nay judgement. It embodies concepts that belong in far more expensive systems. It also manages to be both easy and difficult to use.

If this sounds confusing, perhaps it's just as well. TSM is a rather confusing piece of software. This is not necessarily a complaint. It might turn out to be more of a tribute to the programmers' ingenuity.

The root concept of TSM is that a complete relational database can be constructed entirely using tabular menus and function keys. In TSM you paint your data structure, paint your entry forms, paint your report formats and finally even paint your programs using a number of quirky full screen edit displays. The methodology is idiosyncratic to say the least. The profuse online help often serves only to increase bafflement. However, once you catch on to what the program is trying to do, it usually turns out to be intuitively appealing.

On top of everything else, TSM has two solid factors to recommend it. It's cheap, and it's Canadian. If you want to support the local folks, here's your chance.

Dyna Blow Your Horn

My suggestion for the fledgling TSM user would be to read the manual from cover to cover like a novel. Skipping pages is an invitation to mental disaster, most often manifested by a glazing of the eyes and a drooping of the jaw. On the other hand, thorough reading should be rewarded by clear understanding.

Fortunately, the TSM manual has recently been overhauled. The old book was well written but poorly laid out, consisting of over two hundred half size, plastic coil bound pages... a sort of dot matrix no man's land, with only headers and page numbers as navigational aids. The new version has been totally rewritten, taking into account a good deal of customer input. It also features better print quality... via a Fujitsu twenty-four pin printer... and better binding. The four tutorials that were woven into the original text have now been replaced by seven online tutorials that can be used directly from TSM itself.

TSM is one of those programs built on top of the IBM ANSI.SYS terminal driver. As such, the first thing you'll want to do is construct a bootable disk that installs the driver. TSM is not copy protected, and is small enough to leave a decent amount of data space on your second floppy drive.

Running the program TSM.EXE brings up a boxed menu screen. The first two options relate to setup, and deserve your immediate attention. Although most of the defaults can be left in place, you may need to fine tune the display colours... especially if you're using a composite monochrome tube on your colour graphic adapter. The default TSM text is legible, but I found that the underlined text benefited from being replaced by inverse.

After configuring you'll find yourself at a bit of a loss. Familiar menu choices like "create structure" and "add records" are conspicuously absent from the TSM screen. After some diligent perusal of the manual you'll discover that the "documentation editor" will be your jumping off point into data land. In fact this and the "program design editor" is where you'll spend all your time, up until you've built your first complete working database application. This will take a good while, and you'll get no encouragement in the form of intermediate results. Until the whole application is ready, there's virtually no feedback. I abandoned several parts of the construction entirely, only to be shocked much later to see my apparently failed efforts pop up on the screen in perfect working order.

In the documentation editor you will be encouraged to build a menu screen containing the following options:

1. Data Entry
2. Report Data
3. Sort Data
4. Purge Data

Like everything else in TSM, the editor is truly unique. To begin with you can cursor and type anywhere you like on the screen. After that you'll find that the two square bracket keys can select blocks on the screen... one bracket for the upper left corner of your block, the other for the lower right. Once marked, a block can be cut using F4 and pasted using F8. Rectangular areas can also be boxed using F2 and then the hyphen to select single lines and the equals sign to get double lines.

So far, so good. Of course, you might feel all this is a bit futile, since you so far have no idea how to link any of these tantalizing text descriptions to actual data operations. Before you find out, you'll first have a fascinating side trip into the wonderful world of custom online help.

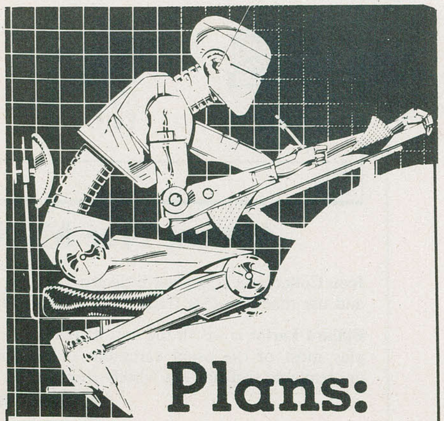
Hitting F3 brings up a messages submenu. The prompts that follow are truly incomprehensible without assistance from the manual. TSM apparently constructs a special data file, with the name *?????MSG.MSG*, where the question marks represent the first five characters of your application filename. The help records are fixed at three hundred and eight characters each, and are keyed using an arbitrary but logical numbering system. You will be prompted to enter help messages directly on your text screen. You supply keys for these messages, but beyond this

they will seem to be ignored by the software, to be left languishing where you left them.

Gritting your teeth, you press on. Returning to the main menu, you now venture into the program design editor. With little warning from the title, you'll find this section of the system demanding the entry of a very conventional file structure. A tabular display allows you to specify field names, a text prompt, sizes, types and display formats... similar to dBASE picture specifications.

After this respite things get really hairy. Selecting the layout option from the design editor screen, you abruptly find yourself in yet another editor space. The upper sixteen lines of this screen are blank, while the lower portion is covered by the most amazing and incomprehensible array of status indicators you've ever seen. There are four lines of inverse headings, each with a line of values immediately beneath in normal video.

At this point I once more came close to complete surrender. Once more the manual came to my rescue. As you move ahead you discover that the many indicator displays are eminently logical, if perhaps a trifle terse. To summarize, the first line



Plans:

Software:	**TSM version 4.0**
System:	**IBM PC**
Application:	**Relational database generator**
Manufacturer:	**DynaBase Superior Software Systems, 51 Tannery Street, Mississauga, Ontario L5M 1V3, (416) 826-5141**
Price:	**$69.95**

shows actions... the equivalent of a program and format description rolled into one. The second line shows the currently selected field from your data structure. The third line is simply static quick reference to your available function key operations. The bottom line contains a series of flags that can often be ignored. Just be careful not to change any of them and you'll find that the default values will get you through. Eventually you can start watching the status line to see a row and column readout as well as a lot of more obscure information.

In operation it all works nicely. You cur-

```
                         Field Dictionary
PROGRAM.$$$ .                      ON-LINE
Field Nbr Identifier         Prompt                 Size Type Display Format
   1      Company Name       Company Name            24   a
   2      Address 1          Address                 24   a
   3      City               City                    24   a
   4      State              State                   3    a
   5      Code               Code                    7    a
   6      Date               Date                    8    a   DD/MM/YY
   7      Company Name       Company Name            24   a
   8      Product 1          Product 1               24   a
   9      Product 2          Product 2               24   a
  10      Product 3          Product 3               24   a

 Fields       Files      Actions    Layouts    ExportOrder  ESCape In/Out
 Report Ctrl  ProgOper   CLEAR      Save       EXIT/QUIT    of this Area

Key | Function                Key | Function               Key | Function
F1  - HELP messages           F7  - Tests actions          DEL - Deletes char
F2  - INC-DEC Subroutine      F8  - Retrieve Sub-File      PgUp- prev 10 defin
F3  - Lists dictionary        F9  - Inserts definition     PgDn- next 10 defin
F4  - Layouts                 F10 - Deletes definition     ARRw- Arrows move
F5  - Copying definitions     INS - Inserts character       [  - Beg Block Mark
Maximum No. Allowed 400       Actual 11                      ]  - End Block Mark
```

sor to a position on the screen, pick a field from your structure by scrolling it into view in the appropriate status line, and then stick it into place using one of four number keys. Number six positions a data entry blank, including the field prompt text and a blank to show the field length. Number two allows you to enter window dressing text. Numbers one and three enter text and numeric fields, respectively, for use on report layouts. As you might surmise from this, the same basic process generates both entry forms and reports.

When you're done painting a decent entry screen, you escape back to the design editor. Instead of your data structure, you now see a similar tabular view of the action statements your painting process has created. You can ignore the meaning of most of this, but you will need to enter at least one action yourself. This one goes at the bottom of the list, and features a Calc/Mode value of seven and a subroutine number of two. Function seven is a record save, and assigning it to subroutine two causes it to save all your entered data every time you finish one record.

Once you've made your way through all this, quit back to the main menu and select the run time command. This will ask you for the name of your program, then pop up that menu screen you painted way back when. Upon hitting the one key, I must say I got quite misty eyed when lo and behold there was my data entry form ready to go. Most remarkable of all, hitting F1... as prompted by a box at the bottom of the screen... reveals the help message you entered earlier.

I guess that's most of the battle, and it would have to get a lot easier with practice. The other menu options... sort, report and purge... are done up in much this same manner, but using different Action statements and a larger number of subroutines.

The Good Word

Beyond merely storing your data, TSM offers all the usual manipulations. Most of these are accessed using truly obscure Type, Xnbr, SubR and Calc/Mode parameters within separate Action statements. The latest release lets TSM programs use chaining and overlays, and allows access to other software from within the TSM system. A library of TSM applications is available: fifteen programs for five hundred dollars.

Aside from transparency, there are several other aspects of TSM that fall short of the ideal. Its file retrieval functions promise but fail to deliver a realistic listing of the available files. Some of these bugs may be missing in the new release. On the more serious side, I accidentally discovered that indiscreet hitting of control keys in the TSM document editor crashes the program.

A major omission is the complete lack of any on line query mode in TSM. The program is not so much a database in itself as a database generator. You have to program it before you can use it. On the other hand, for your trouble you do get a fully relational database. I did not get the impression that TSM is a speed demon, but on flexibility alone it should be able to hold up with the best of them... once you learn how to flex it.

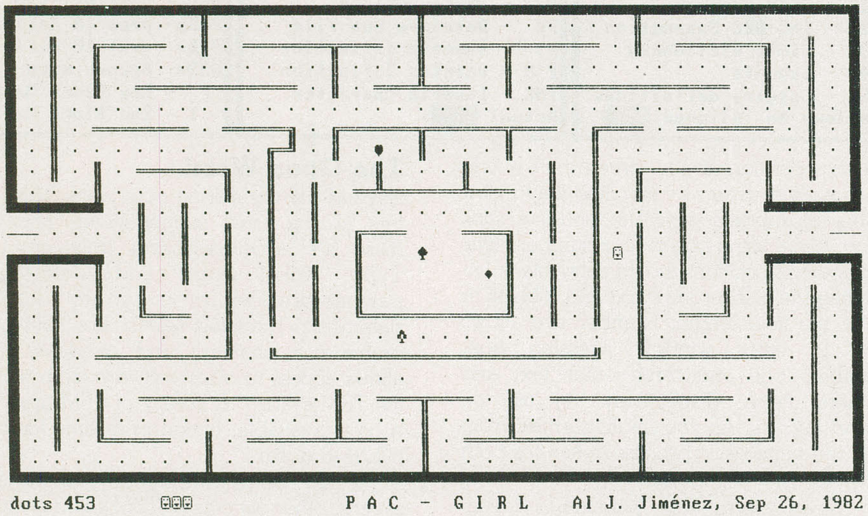
The most remarkable feature of TSM, however, is the price. At about seventy dollars Canadian, with no subsequent distribution royalties on programs written in TSM, anyone developing low cost stand alone applications should find the initial steep learning curve well worth the climb. The ability to swiftly create integrated forms, reports and online help prompts is quite attractive. At what is essentially a shareware price, TSM looks like a worthwhile investment... even if only as an educational experience. **CN!**

Three Almost Fr

for the IBM PC and

This month we really smoked the modem, listening to the wail of carriers long into the night to download the software for these three... count 'em... almost free software disks. There are two here for the IBM PC and one for the Apple Macintosh. All of them are crammed with the cream of the public domain. There are games, utilities, serious things... there was almost a computerized horoscope generator, but it got bumped by the helicopter game.

If you've encountered almost free software before, you may notice that there are fewer titles on these disks than there were on some of our earlier volumes. There is just as much software here... all the disks are within a few kilobytes



Almost Free PC Software Volume 11

Pac Girl is, predictably, a variation on the almost mythical Pacman game. This one moves like a greased cat on a skating rink, with all the speed and generally strange behaviour of the arcade version.

Menu allows one to set up a menu driver, tree structured operating environment that's a great deal friendlier and more manageable than is DOS. It's ideal for creating interactive systems for non-technical users.

Z80MU is one of the most brilliant bits of software we've ever encountered... free or not. It actually emulates a Z80 based computer running CP/M on the PC with no additional hardware... you don't even need a V20. It will run almost all CP/M software, including the old favourites like WordStar and dBase. However, it has features that CP/M never had... and even MS-DOS lacks.

SERIO is the assembler file from the July edition of Computing Now! that implements an interrupt driven terminal in higher level languages, such as C. It's also suitable for use with compiled BASIC. MASM is required to use this code.

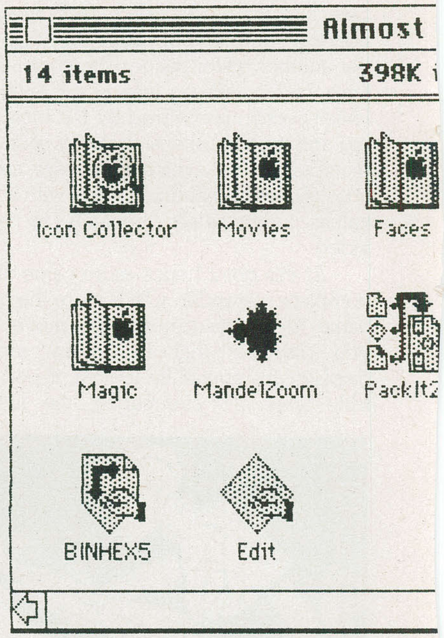
Breakdown is a very peculiar program... it takes meaningful text, analyses it and generates meaningless... but very profound sounding... prose from it. It's fabulous in offices for creating reports if you're wondering if anyone really bothers to read your stuff.

XMODEM is a C language implementation of the XMODEM file transfer protocol, from the July edition of Computing Now!. It can be integrated into other programs to allow one easy access to telecommunications facilities. This code requires SERIO.. above... and version three Lattice C.

GRABIT is the screen grab program from the July Computing Now! It will make a useable text file from the contents of one's screen at the touch of a key. This code requires MASM.

Available for only

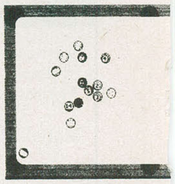
$19.95



Almost Free Macintos

Icon Collector is a peculiar little program that allows y
and use them in other things. It's a surprising amoun

Billiard Parlor is worth the cost of the disk all by itse
play most of the usual variations of pool and billia
unspeakable realism. It's wholly spectacular.



MandelZoom is the nicest Macintosh fractal generator
thing considering the nature of the Mac's floating po

Red Ryder. This is the latest version of this program...
dant rumours of bugs. This one runs perfectly, im
facilities, macros and enough toggles and menus to k

PackIt2...not to be confused with regular PackIt...
downloaded from bulletin boards. It's pretty well esse

BINHEX5 is a file manipulation utility which allows M
Red Ryder.

Edit is the most sophisticated text editor available for t
to edit documents in multiple windows. It produces
things.

Availabl

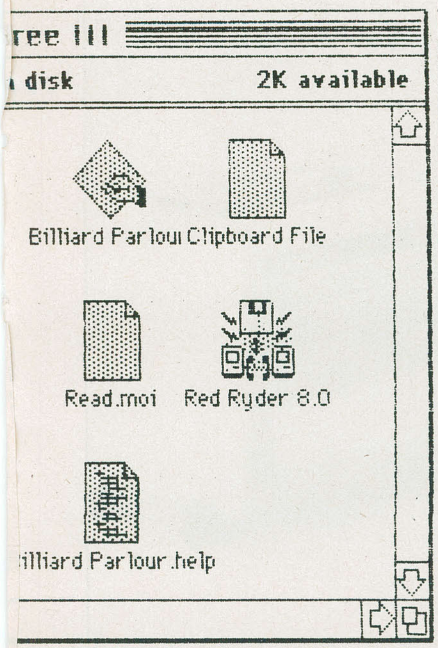
$2

ee Software Disks

the Apple Macintosh

of being full. However, these applications are more sophisticated than some of the earlier programs, and, as such, are bigger.

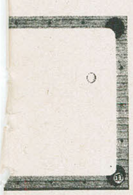
We've had quite a few requests to put the source files from some of the programs we've run in Computing Now! on these disks and, as such, these volumes contain several C and assembler programs. In actual terms, these take up only a few K of code, but they'll save you a lot of typing if you want to play with them.



n Software Volume IV

u to locate icons in applications, swipe 'em, store 'em of fun considering how pointless it sounds.

f. It's a glorious simulation of a billiard table. It will rds, and simulates the movement of the balls with



we've come across. It's extremely fast... a remarkable int library.

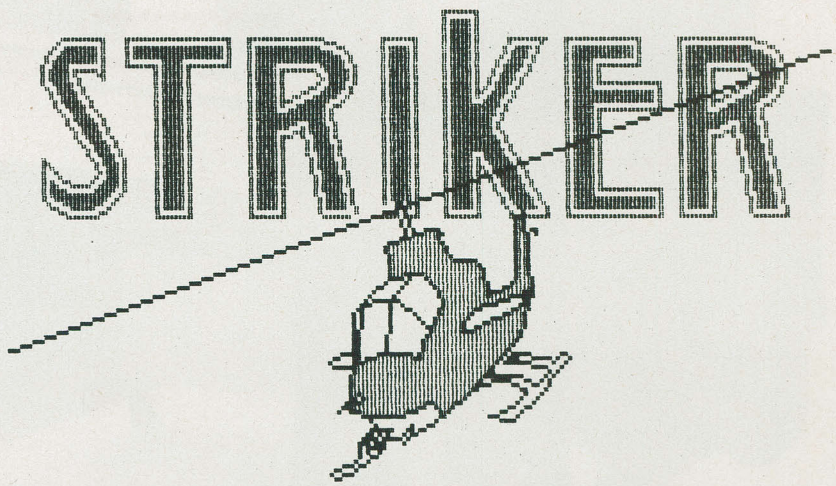
or, at least, the latest one to be released without atten- plementing a sophisticated terminal with download eep your modem in stitches for a month.

will compress and uncompress P2T libraries... as ntial if you're into telecommunications.

ac files to be sent over a modem... another adjunct to

e Mac. Operating similar to MacWrite, it allows one ean text files for use with a compiler, among other

'or only
'5



Almost Free PC Software Volume 12

CV is a small utility to change the volume names on disks. Most humans never think to specify this when creating disks and, thereafter, it's usually unalterable. This is about six hundred bytes of salvation.

Breakout Box is an assembly language program that hides in memory and shows you what your serial ports are doing. It's the most invaluable thing going to solve your serial printer or modem glitches.

Icon Maker allows you to generate sophisticated bit mapped images with relative painlessness. It's easy to use and extremely colourful, producing data that can be incorporated into other programs.

Shell is another DOS menu program. This one is very fast, free of snow, and offers one access to virtually all the DOS features that one might want.

Striker is an experience. It's a brilliantly written helicopter game in the style of Choplifter, complete with professional high resolution graphics and running spies. This is one of the best public domain games we've ever encountered.

Ramset is a RAM expansion program from the July edition of Computing Now!. It allows one to have memory beyond the six hundred and forty kilobyte limit of the PC and to get around the long memory check time associated with lots of RAM.

TRAP is the high resolution Gemini patch program from the May edition of Computing Now!. It makes the Gemini 10x suitable for use with Personal Composer, but is easily modified to fix most bit mapped printing problems. This code requires MASM.

Available for only

$19.95

plus seven percent Ontario disk tax

from

Moorshead Publications
1300 Don Mills Road
Toronto, Ontario
M3B 3M8

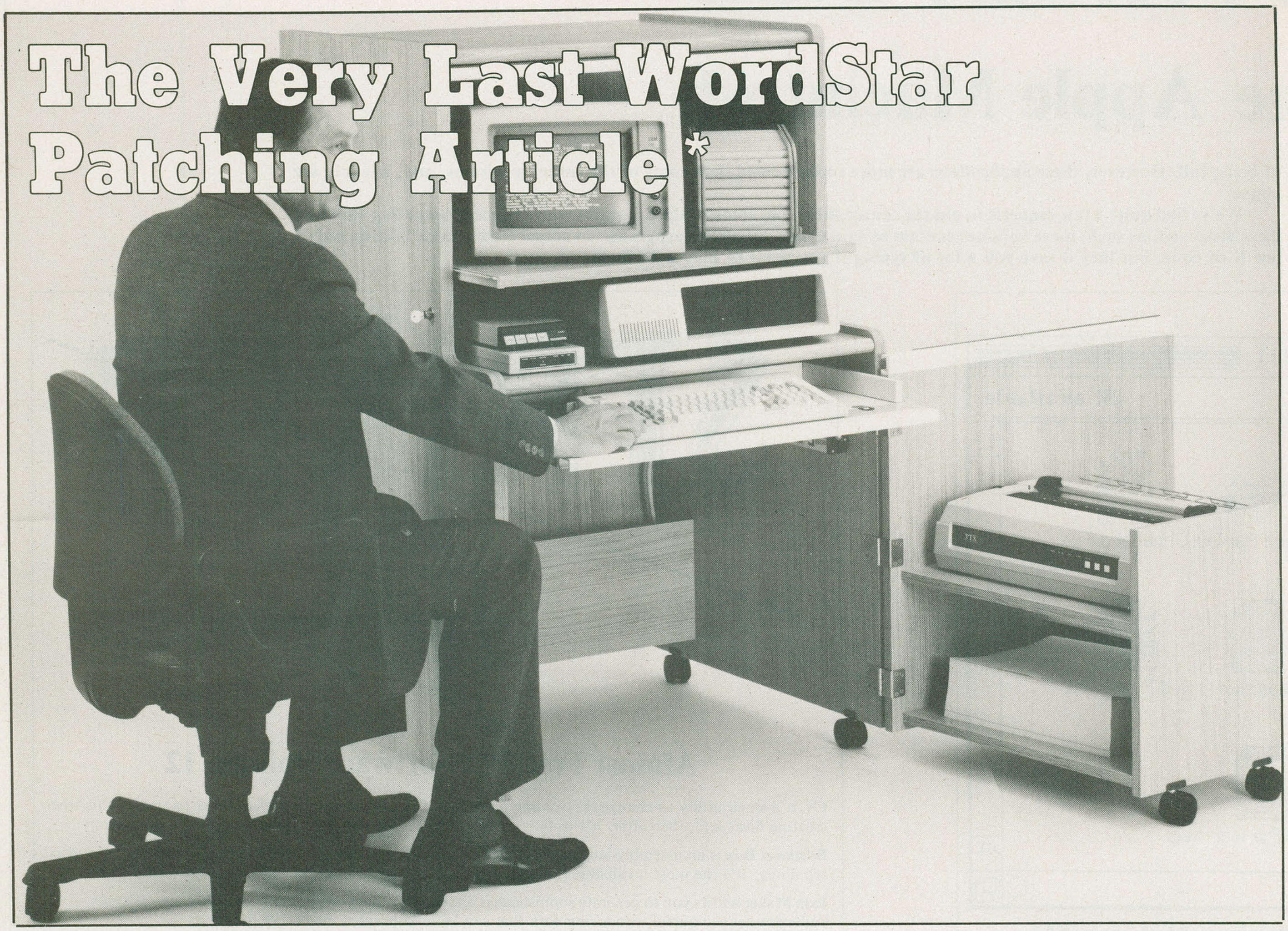
or you can order by phone and frustrate Canada Post. Call

1 (416) 445-5600

Fine print: All of this software was either written by ourselves or collected from public access bulletin boards, and is believed to be in the public domain. All of it has been tested and appears to be functional. However, we cannot assist you in modifying it to suit your individual applications.

We warrant that your disks will be readable. The post office, however, doesn't even warrant that they'll reach you without claw marks. If you are unable to read your disks, please contact us for a prompt free replacement.

The Very Last WordStar Patching Article*



*at least until the next one.

When you get bored of writing with WordStar you can get inside it and hack at its workings. Here's another collection of places to poke.

by Steve Rimmer

Software, like everything else, seems to have a set life upon this world... and when the jig's up, it gets reformatted. At least one needs not bother with expensive funerals, headstones that clutter up otherwise decent fields or lamenting relations. The demise of a program is usually only mourned by its authors, and then only if they haven't sold as many of the things as they'd have liked to.

There are those that would argue that WordStar is one of those chunks of code that is in the last throws of biting the dust. I don't think I'd agree... although it is getting a mite long in the tooth, it still seems to cut it as a well designed... moderately well debugged... general purpose word processor and text editor. In any case, it's easier to keep using WordStar than it is to learn another package of questionably greater utility.

One of the more sublime... if rather more esoteric... joys of WordStar lies in patching it. Because it was written in assembler, rather than in some labyrinthian compiler, tracking through it and hacking at its guts is relatively easy. It can be made to sit up and dance without ever looking at its source code because it's pretty modular and not badly put together.

WordStar hacking is one of the really great relaxations in owning a computer. Undoing some of its shortcomings leaves one with a warm feeling that just falls short of a nuclear meltdown.

In this article we're going to look at a

few more WordStar patches. These are a few of the trickier bits you can lay on it, but they're all easily gotten together with a debugger and a virgin copy of WordStar 3.30. If you have a different release you may have to dig around a bit for some of the addresses we'll be looking at herein.

As always, hack with a copy of WordStar only... not your master... as, if something gets freaked in the process of patching things you can subsequently make another copy.

Functional Code

The function key assignments that come hard wired into WordStar are a bit obscure to say the least. Initially, I adjusted the size of the screen that WordStar thinks it sees to just wipe out the function key line altogether, as it was largely useless and occupied screen space that could have held text. You can do this, if you want to... it's just

```
DEBUG WS.COM
-E248
08DA:0248 18.19
-W
```

You have to type in the nineteen... that's nineteen hex, or twenty-five. This will cause WordStar to print the function key tags on line twenty-six which, of course, doesn't exist.

Killing the function key tags isn't really the most enlightened way to deal with the function keys, however... you can reprogram them for your own uses. Each function key can store a macro of up to six characters. These can be real printable characters or control sequences. You can also change the tags... as we'll see, the actual macro contents and the tags don't really have a lot to do with each other.

There are two rather removed sections of memory to concern one's self with in this process. The most important of these starts at 0670H. If you dump it, you'll see something like this

```
-d670
08DA:0670  02 0B 53 2A 2A 2A 2A 3C-5F 02 0F 47 2A 2A 2A 2A   ..S****<_..G****
08DA:0680  3D 60 03 0F 4C 1B 2A 2A-2A 3E 61 03 0F 52 1B 2A   =`..L.***>a..R.*
08DA:0690  2A 2A 3F 62 02 10 53 2A-2A 2A 2A 40 63 02 10 42   **?b..S****@c..B
08DA:06A0  2A 2A 2A 2A 41 64 02 0B-42 2A 2A 2A 2A 42 65 02   ****Ad..B****Be.
08DA:06B0  0B 4B 2A 2A 2A 2A 43 66-02 11 52 2A 2A 2A 2A 44   .K****Cf..R****D
08DA:06C0  67 02 11 43 2A 2A 2A 2A-47 00 04 11 13 11 05 2A   g..C****G......*
08DA:06D0  2A 48 00 01 05 2A 2A 2A-2A 2A 49 00 01 12 2A 2A   *H...*****I...**
08DA:06E0  2A 2A 2A 4B 00 01 08 2A-2A 2A 2A 2A 4D 00 01 04   ***K...*****M...
```

The first macro string begins at 0670, and is seven bytes long. As with all WordStar strings, the first byte is the actual length of the string, and the next six bytes are the space for it. If you have fewer than six bytes to put in the string, the contents of the remaining part of this buffer are irrelevant, and will be ignored by WordStar. In this case, the unused bytes have been filled in with the byte 2AH, or asterisks, for reasons which remain mysterious.

You can, obviously, change these strings with the debugger and, having done so, write the modified WordStar back to the disk to arrive at a WordStar that will do your bidding when its function keys are prodded. As a sort of obviously visible example, we're going to change the first string, the one that is sent by hitting the first function key.

Looking at the dump above, we can see that the string is two bytes long, and that the first character is 0BH, or control K. The second character is an S... this macro, as it stands, will save one's file whenever it's hit. It would be more useful, however, if it also returned one to one's previous position in the file by sending control K S control Q P. We can make it do this by changing the length byte at 0670 to 04 and adding the other two characters. Change the byte at 0673 to 11H and the byte at 0674 to 50H. If you hit the debugger with a W command, to have it write the changed program back to the disk, you'll have a slightly improved version of WordStar.

I didn't do this with my copy of WordStar. I thought it would be handy to have its function keys print out frequently used words. The word I chose to have the first

Computing Now! Advertisers Index

For advertising information call (416) 445-5600

key print was "wombat". It's a good word and, more to the point, it's only six characters long. You probably won't want to change the function keys of your version of WordStar so they print out words like "wombat". I can understand this. However, we'll look at how it's done here primarily as it gives us an excuse to see how to change the function key label at the bottom of the screen.

This is the dump of the function key string area with the first key having been changed to have it print out "wombat".

```
-d670
08DA:0670  06 77 6F 6D 62 61 74 3C-5F 02 0F 47 2A 2A 2A 2A   .wombat<_..G****
08DA:0680  3D 60 03 0F 4C 1B 2A 2A-2A 3E 61 03 0F 52 1B 2A   =`..L.***>a..R.*
08DA:0690  2A 2A 3F 62 02 10 53 2A-2A 2A 2A 40 63 02 10 42   **?b..S****@c..B
08DA:06A0  2A 2A 2A 2A 41 64 02 0B-42 2A 2A 2A 2A 42 65 02   ****Ad..B****Be.
08DA:06B0  0B 4B 2A 2A 2A 2A 43 66-02 11 52 2A 2A 2A 2A 44   .K****Cf..R****D
08DA:06C0  67 02 11 43 2A 2A 2A 2A-47 00 04 11 13 11 05 2A   g..C****G......*
08DA:06D0  2A 48 00 01 05 2A 2A 2A-2A 2A 49 00 01 12 2A 2A   *H...*****I...**
08DA:06E0  2A 2A 2A 4B 00 01 08 2A-2A 2A 2A 2A 4D 00 01 04   ***K...*****M...
```

If you were to actually make this change to WordStar and save it back to the disk, you would notice that the first function key did actually cause the word "wombat" to be printed to the tube every time it was hit, but that the function key label still said whatever it said before you made the change. This actually makes a lot of sense if you think about it. The function keys really don't usually hold text and, as such, if the labels simply reproduced the actual bytes that the function keys held, they wouldn't avail one of much of a clue about just what the keys were supposed to do in most cases.

As such, each key has a separate label. We can make this label say anything we want about the key... with the restriction that the anything can't be longer than six characters. What is a mite peculiar is the way in which this data is represented within WordStar and where it lives.

You can find the strings starting at 5067H... quite a ways from the place that stores the actual function key data. There are seven characters for each one, the first one being the number that associates the label on the tube with a function key. You don't have to have that number there... you can change it, too, to get a seven character tag. In fact, as far as WordStar is concerned this collection of tags is all one long string. You can change it so that it comes up with your address and postal code if you want it to.

The weird thing about these tags is that they don't exist as straight up ASCII, but, rather, as ASCII characters interleaved with attribute bytes. That is, the first byte... at 5067H... is the actual character that will be displayed, a "1", in this case. The next byte is the attribute in which it will be displayed. If your WordStar, like mine once did, shows its tags in reversed out white type, this byte will be 70H. We'll get to just why in a second.

The next byte along is the next character that will be displayed, followed by its attribute, and so on. This is what this area of WordStar looks like in its virginal form.

```
-d5067
08DA:5067  31-07 48 70 45 70 4C 70 50                        1.HpEpLpP
08DA:5070  70 20 70 20 70 20 07 32-07 49 70 4E 70 44 70 45   p p p .2.IpNpDpE
08DA:5080  70 4E 70 54 70 20 07 33-07 53 70 45 70 54 70 20   pNpTp .3.SpEpTp
08DA:5090  70 4C 70 4D 70 20 07 34-07 53 70 45 70 54 70 20   pLpMp .4.SpEpTp
08DA:50A0  70 52 70 4D 70 20 07 35-07 55 70 4E 70 44 70 4C   pRpMp .5.UpNpDpL
08DA:50B0  70 49 70 4E 70 20 07 36-07 42 70 4C 70 44 70 46   pIpNp .6.BpLpDpF
08DA:50C0  70 43 70 45 70 20 07 37-07 42 70 45 70 47 70 42   pCpEp .7.BpEpGpB
08DA:50D0  70 4C 70 4B 70 20 07 38-07 45 70 4E 70 44 70 42   pLpKp .8.EpNpDpB
08DA:50E0  70 4C 70 4B 70 20 07                              pLpKp .
```

You can change the character bytes very easily with the debugger. Changing the attributes is a bit trickier, as you have to know more or less what you're doing.

Fortunately, this is one of the few times wherein it's a lot easier to be working in hex. The attributes are quite understandable this way. If you think of the hex numbers as being two characters... which is how they're shown... the right hand character is the attribute for the character itself, the foreground, and the left hand character is the attribute for the background.

The attribute 70H, then, means that the character will have an attribute of zero, which is black, and the background will have an attribute of seven, which is white. As such, it will be reversed out.

This is a complete list of the attributes.

0 Black
1 Blue
2 Green
3 Cyan
4 Red
5 Magenta
6 Brown
7 White

If you experiment a bit you'll find that there's somewhat more to it than this. If you choose numbers above seven for the

foreground attribute these colours will show up lighter. Black becomes grey, blue becomes light blue and so on. White becomes high intensity white. If you choose numbers above seven for the background attribute the colours won't change but the characters will flash. I'd recommend avoiding this if you possibly can.

This is a look at the tags area after I've changed the first tag. It will now put up the tag "wombat" in black characters on a red background.

```
-d5067
08DA:5067  31-07 77 40 6F 40 6D 40 62                        1.w@o@m@b
08DA:5070  40 61 40 74 40 20 07 32-07 49 70 4E 70 44 70 45   @a@t@ .2.IpNpDpE
08DA:5080  70 4E 70 54 70 20 07 33-07 53 70 45 70 54 70 20   pNpTp .3.SpEpTp
08DA:5090  70 4C 70 4D 70 20 07 34-07 53 70 45 70 54 70 20   pLpMp .4.SpEpTp
08DA:50A0  70 52 70 4D 70 20 07 35-07 55 70 4E 70 44 70 4C   pRpMp .5.UpNpDpL
08DA:50B0  70 49 70 4E 70 20 07 36-07 42 70 4C 70 44 70 46   pIpNp .6.BpLpDpF
08DA:50C0  70 43 70 45 70 20 07 37-07 42 70 45 70 47 70 42   pCpEp .7.BpEpGpB
08DA:50D0  70 4C 70 4B 70 20 07 38-07 45 70 4E 70 44 70 42   pLpKp .8.EpNpDpB
08DA:50E0  70 4C 70 4B 70 20 07                              pLpKp .
```

More Patches

If your fingers still have a bit of life left in them after you've mutated the function keys of WordStar, you might want to try a couple of other interesting patches. There are a number of potentially useful, if slightly obtuse ones that involve the way in which WordStar relates to its various files.

There are a few operations under which WordStar wants to know what its COM file name is. Specifically, if you use the R option of the no file menu to run another program, WordStar will want to know what program to chain to after your program has run so that it can restart itself. As such, it has to store its own name within itself and, as it doesn't read this when it's started, this string has to be installed within it. It lives at 03E7H.

If you make multiple copies of WordStar for different reasons, each one named something different, you'll have to change this string accordingly if you want to be able to use the R option. For example, the version of WordStar which I've modified to edit program files is called PWS.COM. It was modified, however, from the original WS.COM. In order to use the R option from PWS.COM I would first have had to change the name string at 03E7H to PWS.COM.

Likewise, you might want to change the names of the overlay files. They live at 03F3H for WSMSGS.OVR and 03FFH for WSOVLY1.OVR. There's also the mailmerge overlay at 040CH.

In a more practical sense, if you use a hard drive that comes up as drive C, you've probably noticed that logging onto drive A from within WordStar is a very unpleasant thing to do. All the menus disappear and peculiar error messages generally turn up. The usual recourse in this case is usually to scoot out of the program and ponder for a while.

This will also be the case, by the way, if you run WordStar from a RAM disk.

Actually, this isn't as mysterious as it seems. WordStar is designed to look for its overlay files first on the drive it's logged onto and, if it can't find them, on the drive it has stashed inside it as its default. This lives at 02DCH. Drive A is one, drive B is two and so on. If you change this byte to three it will always look to drive C for its overlays no matter which drive you're logged onto.

The colours that WordStar comes up in can be changed. If you have a colour monitor you can make it quite interesting to look at. If you have a monochrome one you might want to disilluminate it a bit to keep things like your highlighted text from looking unreadable. The attribute colour for the

text lives at 028BH, the colour for highlighted text at 0284H and the colour for the erase colour... what the background looks like... at 02D5H. These bytes work just like the attribute bytes for the function keys, with one digit specifying the foreground colour and the other the background. If you have a monochrome monitor, you might want to change the text byte to 07H... regular white text on black... and the highlighted text with 70H... black text on white. If you don't like looking at inverse video, try using 0FH for the highlight... this will produce high intensity normal text.

Finally, there are a number of delay times built into WordStar that cause it to wait while you're supposedly thinking. As you get better at using WordStar, some of these will want changing. For example, there's a delay that happens if you hit a control character that's intended to bring down a menu. Since the printing of a menu can't be stopped once it starts, hitting, say, control Q and not getting to the next character quickly enough will force you to watch the quick menu... which isn't all that quick... whether you want to or not.

The delay for the menus is located at 02D1. It can range from zero... no delay to speak of... to 0FFH... most of an eternity. It's usually eight. If you increase the value of this byte a bit the menus will think a while longer before cropping up.

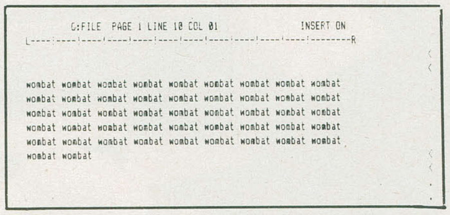
There's also a delay at 02D2H for messages. You might want to lengthen this too.

The delays at 02CFH and 02D0H affect the cursor.

There's Probably More

Actually, I'm sure there's more stuff to mutate in WordStar. WordStar is such a modular bit of programming that it's a party to get inside and meddle with. If you are feeling particularly brave you can get inside it and dig too. All it takes is a debugger and a few pads of paper.

WordStar may be old and funky, but it works. It works a lot better, of course, if you know how to warp its brains a bit. Actually, the hacking of WordStar is probably an art secondary only to its actual use.



CN!

Authors
Scribes
Journalists
Typewriter Repair-persons
Literary Shark Wrestlers
Verbose Pencil Sharpener Technicians
Digital Doorknob Quality Control Inspectors Extraordinaire

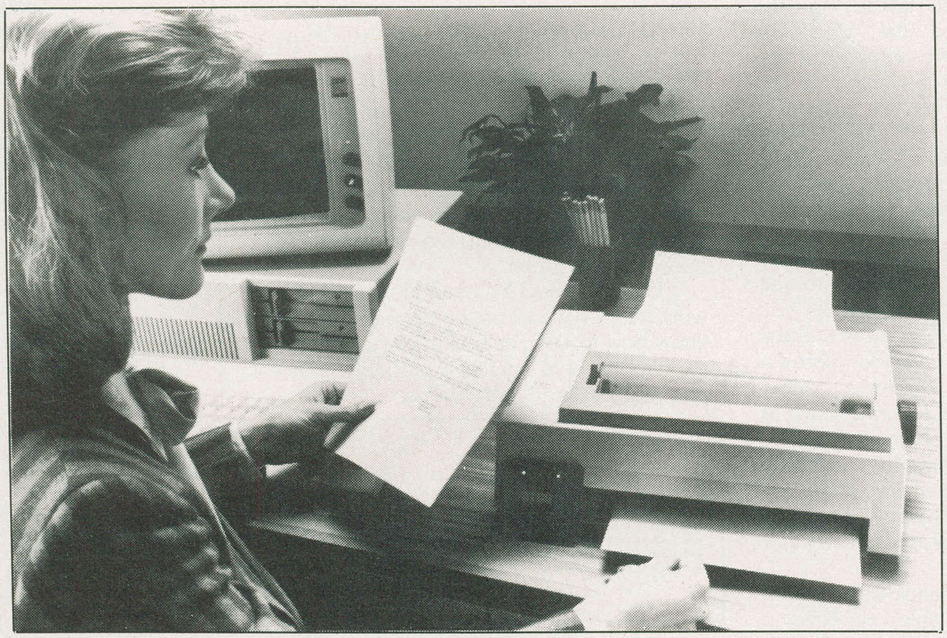
Yes, it's a difficult thing being brilliant and unrecognized. We understand this, and, because we have a profound sense of justice and fair play and quite a lot of paper we at Computing Now! have decided to set forth and *do something about it!* It's all really very simple. Send us splendid articles to put in our magazine. We will send you money and your work will be seen by untold thousands in these hallowed pages.

The only tricky bit is making your articles suitably splendid. We're fairly picky about this condition. However, in order to assist you in this effort, because we are, after all, unspeakably wonderful, we have prepared a set of author's notes. These are useful guidelines that go over things like how to write a good article, how to present it, what we're looking for and, of course, the big one, what we pay. We will also include a list of possibly splendid sample topics to get your brain in gear.

Flog a letter to:

Computing Now! Author's Notes
1300 Don Mills Road, Don Mills,
Toronto, Ontario M3B 3M8

Customizing PC-Write



Perhaps the most sophisticated word processor available for the IBM PC, PC Write is unique in that you can try it before you pay for it.

by Frank Lenk

No piece of software is perfect. If you use any one piece of software at all heavily, chances are you're going to accumulate a long list of complaints, ranging from things like obscure placement of control key functions to indecipherable file formats. Unless you're already an accomplished programmer... with a lot of time on your hands... chances are you just have to grit your teeth and make the best of things.

There are alternatives. In the realm of databases, dBASE succeeds admirably at allowing the user to create his own custom tailored data management systems. On the other hand, if your thing is spreadsheets you'll have to get by with something like Lotus plus a keyboard macro utility.

PC-Write, from Quicksoft, comes very close to being a do it yourself word processor construction kit. The system is so intensely configurable that you can twist it into just about any demented shape you might fancy. Furthermore, PC-Write macro programming lets you just about throw your copy of ProKey out the window. Customization extends far beyond the keyboard, allowing you to do weird things to your screen and printer output as well.

As you *should* know by now, PC-Write can be had free for the asking, although you'll eventually want to consider splurging the voluntary registration fee of seventy-five dollars... American... to qualify for updates. Author Bob Wallace has just recently graduated his brainchild from version 2.55 to version 2.6, adding a whole new three ring circus of features. There's also a new manual... a twenty dollar purchase for previously registered PC-Writers.

What follows is a quick look at some of the tricks we've picked up over several months of daily PC-Writing. This should help the beginner get some grasp of what's going on. If you're still holding off giving PC-Write a try... heaven knows why... this may give you a hint of what a party you're missing.

Letters To The Editor

My own copy of PC-Write is so thoroughly hacked these days that no other knowledgeable user could possibly puzzle it out. I've relocated most of the key functions, modified some that I wasn't crazy about originally and added a few peculiar ones of my own. I've also done a few obscure things by way of fine tuning my printer output.

All this took quite a while to accomplish, and a good bit of acquaintance with the software. The good news is that to get started you really need to know very little.

In the new version 2.6, you need to know less than ever. The new PC-Write goes to the trouble of emulating almost all the old faithful two key WordStar commands. Anyone making the conversion should find it immensely comforting that all the control K block functions, control Q search and control ESDX cursor diamond operations are still available. Left to itself, however, PC-Write has an inclination to use the function and cursor keys for editing functions. The WordStar keys mimic only a handful of the staggering number available.

Practically every program ever written incorporates some sort of configuration options. What this usually amounts to is telling the system about your default disk drive and whether you have a colour monitor.

In PC-Write the approach is rather different. Customization is ingrained right into the fabric of the program. All functions appear to have been vectored through in such a way that new values can easily be read in. Whenever the main ED program is invoked it automatically looks around for control files that can contain alternative values for each command vector. While they're at it, the control files can also provide on the fly information about things like screen colours, system configuration, printer setup, formatting and more.

If you ignore the control files entirely, PC-Write simply defaults to acting like any other preset application environment. However, even the complete novice can take advantage of the simple starter set of control files provided on the PC-Write distribution disk. As you gain mastery over all the options, the default files tend to ex-

pand rapidly... and PC-Write tends to look more and more like your ideal word processor.

PC-Write consists of two separate programs, ED.EXE and PR.EXE... one being the editor itself, the other the printer module. Each of these two programs takes its own configuration control file. These are logically dubbed ED.DEF and PR.DEF, and are read automatically when their respective program files are invoked.

Actually, even this level of operation is accessible to user control. The first control command we come across is of the form

!ED.*

Appearing somewhere in the ED.DEF file, the exclamation point command would cause PC-Write... after loading ED.DEF itself... to search for another control file with the name ED and an extension matching the extension of the file being edited.

By using consistent filename conventions... such as .LET for letters, .TAB for tabular material, or whatever you like... you can have PC-Write load specialized control files for different types of tasks. Letter writing commands not required for tabular entry can be replaced by something more useful. Formatting can be completely reset to match the type of document.

Actually, any filename and extension could be substituted in the exclamation command line, but ED has the advantage of following the original PC-Write control file nomenclature. There can be up to six exclamation commands in the .DEF file. A common option would be to include one like

!PR.DEF

This would cause the printer control file to be loaded even when editing. Often this file will contain formatting information... for instance, page length... that could be worth having on tap during the edit process.

Control files can have several types of command lines, preceded by the following characters

! (control file spec)
% (system configuration)
& (display configuration)
 # (printer code setup)
@ (proportional font data)
$ (printer configuration)
nnn:nnn,nnn,"string" (key definition)

Keyboard redefinition lines consist of a three digit key code followed by a colon and the assigned sequence of key codes, separated by commas.

The percent sign precedes single letter function codes, each of which sets some configuration option. For instance, A indicates that the program is to use a monochrome display adapter. Other codes toggle colour graphics snow off and on, enable or disable menus, set up for use with a PCjr keyboard, toggle auto reformatting off or on, and so on. The S code is particularly interesting, setting "sticky" mode for shift, control and alt keys. This allows two key codes to be entered one key at a time, useful for those who can't... or won't... use both hands on the keyboard.

Modifying the PC-Write keyboard is actually one of the simplest configuration tricks. All you need is one control line for each key to redefine.

Key definitions look like this:

339:261 (backspace)
010:327 (control-J)
011:335 (control-K)
485:"Name?" (shift-control-n)

The first of these simply defines the *del* key to mean *backspace*. The 339 represents the normal code for del, and 261 is the code for backspace. The colon does the assignment. Anything in brackets is treated as a comment and has no effect on configuration.

The next two lines are the definitions I use to make control J and control K take the cursor to either end of the current line. This loses access to the WordStar block commands, but I find it a fair trade. There are so many functions available in PC-Write that you find yourself making this kind of trade off rather frequently.

The final definition simply assigns the quoted string to the code for control N.

When entering key codes you have two choices. You can simply look them up in the documentation, or you can use the numbers mode. Hitting control caret... control and the number six key... causes all further keystrokes to enter their numeric code rather than their usual function or ASCII value. Hitting control six again gets you back to normal typing. This procedure is not quite as easy as the co-resident editor of ProKey, but then again PC-Write makes it surprisingly easy to switch over and edit your control file, save it, implement it on the current environment, and then return to your original edit file.

Edit functions have been assigned arbitrary code numbers... roughly in the order Bob Wallace added them to the program, I would imagine. There's no easy way of accessing these via the numbers mode, so you'll have to look them up. However, you don't want to be changing fundamentals all that often anyway.

If you do start browsing through the manual, you'll find that the selection of functions is staggering. Code 307... normally assigned to shift keypad 5... automatically strips WordStar high bits from the file in memory. Code 400 is an "autoexec" function, assumed to be pressed once every time PC-Write starts up... very handy for initializations. Codes 409 and 410 do a block mark on the next word or the next line, respectively. These latter two useful little functions highlight the potential of the software: by default neither of them is assigned to a keyboard code in the latest version of PC-Write. There's just nowhere to put them.

In fact, PC-Write uses a fantastic array of key combinations to access as many functions as possible. Almost every key on the keyboard has a meaning in unshifted, *control*, *shift*, *alt*, *shift-control*, and *alt-shift* variants. For instance, in version 2.6 alt and shift or alt and control together with the letter keys will produce the various PC box drawing characters. Alt together with the letter keys is used to access preset printer escape code sequences to get underlining, boldface and other print effects.

This doesn't have to be too mind boggling, if you go slow. You can start by doing small fixes on the original PC-Write logic. For instance, I found that the use of F4 as the block delete key placed that dangerous function a wee bit too close to my wandering pinky. One of my first redefinitions was to swap the F4 operations with those of F3, normally the block copy key.

Later I started creating my own compound functions, just as one might with a co-resident macro utility. For instance, the following line makes *alt F8* centre the cursor between the margins, without entering any text.

367:046,347,328,261 (aF8 ctr)

This sequence prints a period on the screen, centres it, then backspaces it out of existence. It's handy for positioning the "Sincerely yours" on letters.

Putting On A Display

Setting up a printer is one of those tedious, obscure sorts of jobs that few users probably ever get quite right. The folks at Quicksoft have anticipated this, and provide direct support for a phenomenal number of popular printers.

A program called MENUPRT, supplied on the PC-Write distribution disk, offers a menu of printer makes, then a submenu of models for the chosen make. Finally it churns out a control file containing all the basic codes required to allow PC-Write to properly drive your printer. The starter file I got for my Panasonic 1091 had a couple of minor problems that needed patching up, but I'm still using it fairly much in its original form.

The number sign commands form the core of the printer control file. A typical line might be as follows.

#B=02 +27,69 -27,70
#C=06 @17i +15 -18
#D=16 +27,87,1 -27,87,0
#E=03 +27,77 -R

Each line starts by specifying what letter... together with the *alt* key... will initiate the font in your text. Thus the first line specifies *alt B*. The two digit number after the equals sign is the ASCII code of the character that will be shown in your text as a marker.

These marks can be hidden or displayed by hitting *alt spacebar*.

The ASCII codes following the plus sign are sent to the printer the first time the code mark is encountered in your text. The codes after the minus sign are sent when the second, matching code is reached. Notice the prevalence of twenty-seven... the ASCII code for *escape*, recognized by most printers as the start of a command sequence.

The second line above defines *alt C* as the start of compressed printing. The at sign in the file specifies an alternate character width to be taken into account when formatting text marked with this code. The fourth line shows the coding for elite pitch printing. An R following the minus sends an appropriate code to resume printing in the font and pitch active prior to the switch to elite. This lets "nest" your font changes without getting lost.

The at sign can be used at the start of a command line to specify character width tables for proportional printing. MENUPRT will insert tables in your control file if your printer has any use for them. Setting up your own table would be tedious to say the least, but at least the option is there.

Although the above examples don't show it, command lines can also include specification for overprinting. Thus you could redefine the IBM's extended foreign language accented characters to print as a normal ASCII letter with an overstruck apostrophe or quote mark. These characters would then print transparently even on non-IBM printers. Anyone who's ever tried to install WordStar to take advantage of a decent dot matrix printer should begin to appreciate the beauties of programmable printing.

Dollar sign command lines in the printer control file act much like the percent editor commands, setting things up to match your hardware. For instance, $P specifies an initialization sequence to be sent when you start to print. Other codes let you specify line termination with or without line feed and carriage return... an important consideration on some printers.

Macroscopic formatting is handled throughout by "dot commands" based on those found in WordStar. However, the new release of PC-Write insists on seeing an *alt G* male symbol character in front of the dot... presumably to permit faster processing. The command *control backslash* has been added to easily add the extra code to all dot commands in your existing files.

The editor control file will generally contain a ruler line, also marked with the *alt G* code. This defines the positions of margins and tab stops, and sets justification mode. The last ruler line read takes precedence, which lets the exclamation point command call up an appropriate ruler file according to its extension. For instance, editing a file with the extension .LET might call up a special control file containing nothing more than a new set of margins, and setting justification for ragged right... appropriate for correspondence. Having just a ruler in the special .LET file would leave all the ED.DEF key definitions intact. Of course, you could monkey with those as well.

The ruler is just a special case of the general dot command. Dot commands set headers, footers, footnotes and all the other fancy formatting stuff. Rulers and other dot lines can be embedded anywhere in your control files or in your text. PC-Write version 2.6 provides special commands to let you pick up, drop or store away your custom rulers. That means that you can now have all sorts of multiple formats in one file.

Ampersands are used with the numbers from one to fifteen, controlling the display colour of various PC-Write features. For instance, &1 sets the colour of normal text, and &5 sets the colour of text in a marked block. Ampersand lines have the form:

```
&n: nnn.nnn.nnn
```

The first *n* represents the feature code. The other three groups control colours for monochrome, single colour composite and full colour RGB displays. This single compact command can thus set up the software to work on various machines... great if you have slightly different systems at home and in the office.

Furthermore, each *alt* printer font code can have its own display spec, embedded in the appropriate number sign command line. The colours are defined using the same attribute codes as above. All these options make PC-Write particularly easy on the eyes, no matter what kind of monitor and display card you happen to be using.

Help Is On The Way

If all this begins to get the better of you, never fear. In PC-Write, help is just a stab of the F1 key away. Actually, in version 2.6 it usually takes two stabs, but compensation is that you can *always* get to the help screen by bashing away at F1 long enough... no matter what mess you've gotten yourself into. Of course, if you insist on reassigning the command function of F1, you're on your own.

The PC-Write help file... ED.HLP, in its latest rendition... is a separate file on your startup disk. The interesting thing about this file is that it is a totally standard PC-Write text file. You should feel free to suck it into PC-Write and chew it up thoroughly. You can even use printer codes to highlight text and generally dress up your displays.

The help menu occupies the top half of the help screen, and consists of a large number of topic headings arranged in a rectangular grid. This grid appears at the top of the help text file. You can change any of the headings, even change the total number of headings, as long as your table continues to be exactly eighty columns wide and as long as each topic title starts with a capital letter.

Help pages are marked off with page break codes. Running through the heading menu... using the cursor keys... will display the help pages, in sequence.

In version 2.6 the help file has grown to about forty kilobytes. However, if you cut it back to under twelve you can still take advantage of a trick used in the earlier versions. Placing a "I" line in your configuration file will cause PC-Write to stash the help text in the portion of video display memory that normally goes unused in text mode. The gain in RAM space can be important if you like to have lots of co-resident stuff on hand.

If you really get lost, there is a last resort. Shell out your seventy-five bucks... about a hundred Canadian beaver skins... and get registered. Then you can call Quicksoft directly. I've tried them two or three times on phone support. At least once they didn't know what I was talking about... but they did try to figure it out. The problem turned out to be with another program I was trying to hook in. At least one other time the person I spoke to was already aware of the program bug that was causing my problem. She gave me a simple bandaid solution over the phone, and the recent upgrade to version 2.6 has smoothed the whole thing over.

It seemed worthwhile to dwell in some detail on the programmability of PC-Write. However, to do this I've had to go very light on some of its other qualities. PC-Write has always been about the speediest word processor made for MS-DOS. It has also been one of the very few that is equally good at handling formatted text and pure ASCII program code. Version 2.6 has added a huge number of refinements, including automatic paragraph reformatting, embedded format lines, proportional printing support, expanded help, optional Lotus type menus and a whole lot more.

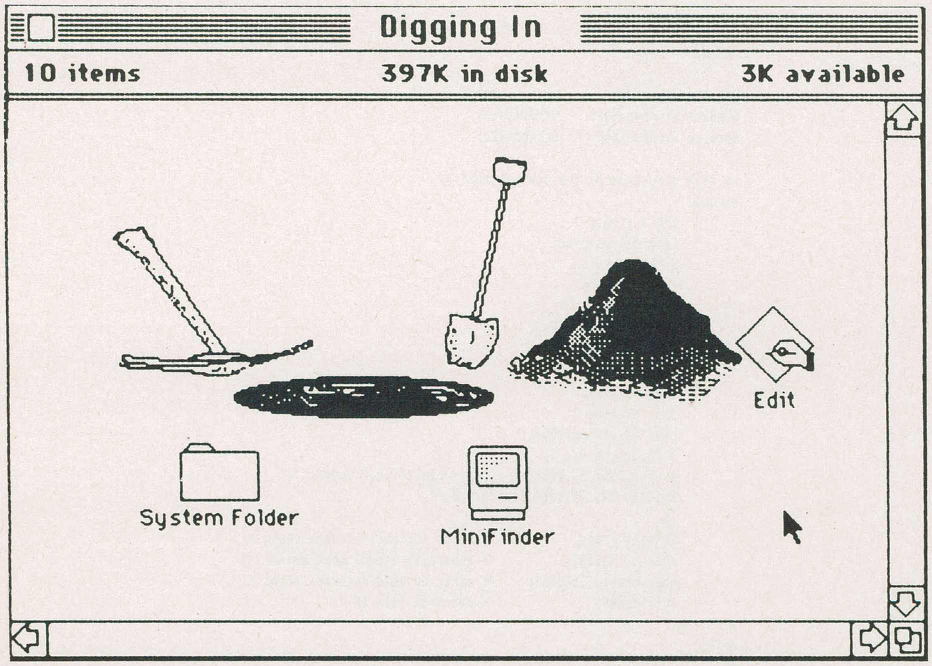
PC-Write files are still limited to sixty kilobytes... an adequate amount for most mortals... and page breaks are still not dynamically displayed. Ah well... I'll wager these are both tricky things to rectify without impairing the program's present enviable level of performance.

The new release has, however, included the first really professional PC-Write manual. The old book was fun reading, but a bit impenetrable for the novice. The new manual is clearly laid out, well written and attractively printed. **CN!**

Digging Into The Mac

While by no means easy, the Macintosh is an authentically rewarding box to write code for... once you get over its countless peculiarities, bizarreties, weirdnesses, inconsistencies, discrepancies, ambiguities, unusual phenomena, conceptual hangups and references to Pascal. This feature takes you past the initial tricky bits.

by Steve Rimmer



I think that the Macintosh has about as much future in business as do the fifty hour work week, fuel injected swivel chairs and old fashioned craftsmanship. However, it's a splendid computer for having fun with and... once you get around its several hundred internal quirks... a surprising and interesting box to program. Those quirks, though... they'll do your head in if you're not watching for them.

The operating system of the Mac is unlike that of almost anything that came before it. All operating systems provide one with some internal resources but, whereas, say, MS-DOS is good for a few dozen system calls, the Mac has well over four hundred and fifty of the little monsters, with new ones cropping up every so often. Furthermore, whereas one communicates with the operating systems of lesser computers by loading up a few registers, the Mac wants to be passed pointers to oftentimes gargantuan data structures. One of the most daunting aspects of the Mac is the size of the task one faces in writing even simple code for it.

Making something actually go on a Macintosh, then, is rather a hoot. It makes you feel really good when you finally get your head around the cryptic documentation, translate all the references from Pascal into whatever it is that you're using and really get an application happening.

In this feature we're going to look at the structure of a Macintosh application written in DeSmet C, a typical programming language for the system. We're also going to look at a small program... small being a relative term... to see how some of the more common special effects are gotten together in a Mac.

Natural Resources

One of the first things that one realizes about writing code for the Mac is that even a trivial application will be huge. It's virtually impossible to write anything worth having in fewer than a thousand lines of code. I decided to try to keep this example program down a more manageable size, about four hundred lines, to make it typeable in the lifetimes of any souls wishing to enter it. As such, it's to a large extent incomplete and somewhat functionless.

It does serve, however, to illustrate a number of the precepts of Mac programming. It's also mildly interesting to play with.

The program is a "resource peeker". Run it and you'll be able to open files that have resources attached to them and check things out. The program will list the resources available in the file and tell you briefly about each one. In the case of ICON resources, it will draw the icons.

More to the point, the program uses a lot of the visual things we've come to recognize as being part of the Mac's internal works. It has an "about" box, it opens lots of windows, invokes the get file package... a specialized dialog... handles the mouse, does a custom made menu and uses a few of the more specialized resource management routines. It also gets the whole effort together without throwing system errors.

If you want to you can make this thing into a complete resource editor, although there's little reason to want to do so, as there are already several in the public domain. Check out our Almost Free Macintosh Software volumes one and two. Perhaps more practical, you can use this code as a beginning for your own programs.

There are a number of things worth mentioning before we get really tight with the code itself. The Macintosh's operating system was written in Pascal, which is an interesting little language, to be sure, but a bit funky to actually write in for most heads. This is understandable... the fellow who devised it never really planned to have it used for programming. I find C a great deal easier to use... in addition, there are some really superb C development packages extant in the galaxy.

This program was written with DeSmet C for the Mac, which is by no means the most sophisticated compiler available. However, it has several things to say in its favour. It's not badly documented, maintains a very complete Mac interface, it's quite a bit faster than several other efforts

and, perhaps most important, it's easy to use. It's also reasonably inexpensive. There are a number of other compilers, such as Aztec C and Consular C which are equally useful.

The fact that the Mac was written in Pascal also implies that it wants to run programs which were written in Pascal, or, at the very least, look like they were. Pascal has its own internal representation for things like strings and complex numbers which... not surprisingly... differs from that of C. The compiler will make most of the translations for you. There is one instance in this example, however, wherein the switch has to be done by hand.

The documentation for the Macintosh was also written in Pascal or, more to the point, was intended for people programming in Pascal. You may have a bit of a dance with it in trying to program in C, but you can get around it once you get used to the notation. The principal work, *Inside Macintosh*, used to be published by Apple as two really funky loose leaf binders full of badly photocopied notes. It has just recently been done as a huge hard covered book by Addison-Wesley which has most of the bugs fixed and everything in a much more useful form. It costs well over a hundred bucks, but if you're even thinking about programming the Mac you'll pretty well have to score a copy.

Sadly, this isn't the sort of book to be remaindered at Coles for a dollar ninety-nine if you wait long enough.

The last thing that's worth mentioning in this preamble is a bit about what this program does... and why. In fact, its structural bits are far more important than its function... you will most likely want to use its component parts in programs of your own rather than get into this thing in any heavy depth. However, understanding just what it's about may make its pieces more useful.

Unlike, say MS-DOS, the Mac stores its files as two bits, these being the data fork and the resource fork. Some files have only one or the other. A program source file, such as this one, would have several kilobytes in its data fork and nothing in its resource fork. A program... such as the one this source file will generate... will have a big resource fork and little or nothing in its data fork.

The data fork holds information, text, bit patterns and so on, while the resource fork holds... well, it holds what the Mac calls "resources". A resource is anything that can be used as part of the execution of a program. This includes actual code, of course, and it divides this into quite a variety of types of routines, but it also includes things like fonts, icons, patterns and so on.

Every resource type has a name, such as ALRT, ICON, FONT and so on, and each name can encompass several actual resources, each with a unique ID number.

```
/*
:       copyright (c) 1986                      :
:       steve rimmer                            :
:       in deSmet C                             :
:                                               :
:       resource peeker:                        :
:       the ultimate application.               :
:       serves equally well as a                :
:       word processor, spead sheet             :
:       and data base manager                   :
*/

#include        <quickdraw.h>
#include        <window.h>
#include        <menu.h>
#include        <event.h>
#include        <packages.h>
#include        <textedit.h>
#include        <dialog.h>
#include        <stdio.h>

MenuHandle menus[10];

/*      constants for the menus         */
#define apple menu      1
#define file_menu       2

/*      constants for the mac   */
#define ScreenWide      512     /* number of pixels across the screen */
#define ScreenDeep      342     /* number of pixels down the screen */

WindowPeek curwindow;
TEHandle hTE;

Point nextwindow        =       {40,4};
#define windowheight    ScreenDeep;
#define windowwidth     ScreenWide;

/* this struct holds the dialog stuff */
struct{
        int d_items;
        long d1_reserved;
        Rect d1_rect;
        char d1_type;
        char d1_len;
        char d1_string[8];
        long d2_reserved;
        Rect d2_rect;
        char d2_type;
        char d2_len;
        char d2_string[128];
        } ThingList = { 1,
        0L,{64,125, 82,200},ctrlItem + btnCtrl,8," Cosmic ",
        0L,{10, 50, 24,150},statText,0,"",
        };
        SFReply Frep;           /* reply for get filename thing */
        char scrap[64];         /* generally useful text buffer */
        char resName[64][5];    /* array to hold resource names */
        int ResRef;             /* resource file ID */

main()
{
        WindowPtr mywindow, NewWindow();
        Rect wR;

        InitGraf(&thePort);     /* initialize stuff */
        InitFonts();
        InitWindows();
        InitMenus();
        TEInit();
        InitDialogs(0L);
        InitCursor();
        InitAllPacks();

        FlushEvents(everyEvent);        /* kill any pending events */

        curwindow = 0L;
        hTE = 0L;

        GrafPort myport;                /* pop open a grafport */
        OpenPort(&myport);
```

For example, if one opens the system as a resource file, selects the ICON resource type and takes resource ID one, one would have a handle to an icon. If we were to plot this icon we'd see one of the alert symbols.

There are pots of routines to handle all this resource juggling. We'll look at some of them here, but resource management on the Mac is a somewhat gargantuan topic.

Crack The Window

Unlike as with most C programs, C on the Mac requires quite a few initial mind games before one can get down to the good stuff. In fact, in most cases, programs under the Mac don't behave quite the way you'd expect them to.

The first thing one does upon typing *main( )* and getting the whole effort under way is to initialize any of the Mac's internal bits one plans to use. In fact, this process is extremely fast, and I usually just initialize everything in sight. It reduces the possibility of trying to use something that hasn't been initialized later on... this can see the Mac generate some of the weirdest errors.

The next thing to get together is the opening of a big window to work in. Since we'll be opening a number of windows in the course of this code, let's check out the mechanism here.

The most obvious thing about a window is its size. A window is defined by four coordinates passed to the window manager as a data structure of the type *Rect*. This is actually a struct having four *ints* called top, left, bottom and right in that order.

The first thing to do, then, in opening a window is to declare a variable of the type *Rect* and assign its elements the coordinates of the window.

There are several other arguments for *NewWindow( )*, the Mac routine that gets windows together. Some of them aren't usually important, and we won't talk about what they do here. The usual form is

NewWindow(0L,&rect,"Window Name",1,DocProc,-1L,1,0L);

The *rect* is, of course, the structure that defines the location of the window. The string immediately after it will show up as the title of the window if it has one. Some types of windows don't, in which case you can pass a null string. The contents of a non-null string would be ignored in this case. The field that I've filled with *DocProc* is the window type. There are a number of these... check out *Inside Macintosh* to see what they're supposed to look like and the type codes that generate them.

The *NewWindow( )* procedure returns a window pointer of the type *WindowPtr*. It doesn't really matter what this looks like internally, so long as we use it properly. Specifically, we'll want to hang onto this so that we can pass it to *DisposeWindow( )* when it's time to close up shop.

```
                wR.top = 20;
                wR.left = 0;
                wR.bottom = ScreenDeep;
                wR.right = ScreenWide;          /* and a principal window */

                mywindow = NewWindow(0L,&wR,"Peeker",1,noGrowDocProc,-1L,1,0L);

                InstallMenuBar();               /* some menus, Gaston */

                GrabEvents();                   /* main program */
                DisposeWindow(mywindow);        /* waste the main window */
        }

        GetFile()               /* open a file */
        {
                long f;
                Point a;
                char n[128];

                a.h = 100;
                a.v = 50;
                SFGetFile(&a,"",0L,-1,&f,0L,&Frep);     /* do the box */
                if (Frep.SFgood) {
                        if (strcmp(Frep.SFfName,"System") != 0) {
                                SetVol(0L,Frep.SFvRefNum);
                                if((ResRef = OpenResFile(Frep.SFfName)) != -1) {
                                        UseResFile(ResRef);
                                DisableItem(menus[file_menu],1);
                                EnableItem(menus[file_menu],2);
                                EnableItem(menus[file_menu],3);
                        }
                        else {
                        sprintf(scrap,"Can't open %s",Frep.SFfName);
                        ErrorAlert(scrap);
                        }
                }
                else {
                        ErrorAlert("Can't open system");
                }
        }
}

ErrorAlert(s)           /* a simple dialog box to say something's amiss */
        char *s;
{
        DialogPtr d, NewDialog();
        Handle i, h, NewHandle();
        Rect r;
        int t,p;
        char f[128];

        ThingList.d2_len = strlen(s);
        strcpy(ThingList.d2_string,s);

        i = NewHandle((long)sizeof(ThingList));
        _move(sizeof(ThingList),&ThingList,*i);

        r.top = 75;
        r.left = 100;
        r.bottom = 160;
        r.right = 350;
        d = NewDialog(0L,&r,"",1,dBoxProc,-1L,0,0L,i);

        do {
                ModalDialog(0L,&p);
        } while (p != 1);
        DisposDialog(d);
}

ShowAll()               /* list all the available resources */
{
        int l,i,j;
        int x,y;
        char m[4];
        WindowPtr n, NewWindow();
        Rect r;
        struct mouse { short y,x;} p;

        l = CountTypes();               /* how many are there? */

        r.top = 50;
        r.left = 10;
```

The next thing we encounter in the main part of the program is the *InstallMenuBar( )* function, which is one of mine. You can find it further down in the code. It sets up the menus at the top of the screen.

The menus that the Mac provides are considerably more versatile than most applications software would give you cause to imagine. There is a lot more that you can do with them than most programmers get into. We're not going to check out all the freaky details here. However, a peer at the *InstallMenuBar( )* routine will give you an idea how this all works.

The array *menus* is actually an array of menu handles. In a more complex program there would be quite a few more of these. Because these things are declared globally, we can get at them in other functions. This is important, as the *EnableItem( )* and *DisableItem( )* Mac calls have to know which menu handles to work with. There are similar routines which install check marks and so on dynamically in menus... I haven't used them here.

The process of adding things to the menus should be pretty clear. The *NewMenu( )* function scraps the current contents of the menu and returns a handle to the menu space. The *AppendMenu( )* call places the actual text and the *DrawMenuBar( )*function does largely what you'd expect.

The text passed to *AppendMenu( )* is a bit tricky. The items are obviously separated by semicolons. You can cause an item to come up disabled by putting an opening parenthesis in there with it. An exclamation point followed by a character constant will cause the item to be flagged by that character.

The main beasty is the *GrabEvents( )* function, which is a *while* loop to trap anything the Mac senses as coming from the outside world. In our case, this will be wholly confined to mouse clicks in various areas of the tube, but it could include keyboard activity and other phenomena.

The places we're interested in looking for mouse clicks are in the menu bar and everywhere else. The system splits these up for us very conveniently. The *GetNextEvent( )*function returns an event record which contains, among other things, the nature of the event that has come down and which window it has occurred in. We can switch on the value of *myevent.where* to see where the mouse has been belted.

The *menu_command( )* function actually calls the code that does something as a result of a menu's being selected. In fact, there are only two menus in this program, but the structure of this thing should be fairly easy to understand, and can be extended into whatever size fuming mess you're up for. One of the things that switches do to one in larger programs is to get so big as to

```
        r.bottom = 175;
        r.right = 300;
        n = NewWindow(0L,&r,"Resources",1,dBoxProc,-1L,1,0L);

        if (n != 0) {
                SetPort(n);
                for (i = 1; i <= l ;i++) {
                        GetIndType(&m,i);           /* get one */
                        x = (((i-1) / 10) * 48) + 10;
                        y = (((i-1) % 10) * 12) + 10;
                        MoveTo(x,y);                /* show its name */
                        sprintf(scrap,"%4s",&m);
                        DrawString(scrap);
                        strcpy(resName[i-1],scrap); /* save it in the array */
                }
                WaitForRat();                       /* wait for click */
                GetMouse(&p);                       /* where was it clicked? */
                x = 48 * (p.x / 48);                /* get column and row */
                y = 12 * (p.y / 12);
                i = ((x / 48) * 10) + (y / 12);
                if (i <= l) {
                        r.left = x + 10;
                        r.right = r.left + 48;
                        r.top = y - 1 ;
                        r.bottom = y + 12;
                        InvertRect(&r);             /* momentarily invert the */
                        WaitForNoRat();             /* appropriate resource name */
                        InvertRect(&r);
                        SeeResource(i);             /* and see what it's about */
                        DisposeWindow(n);           /* close the window */
                }
        }
}

SeeResource(a)  /* show the resource type with name in array element a */
        int a;
{
        WindowPtr n, NewWindow();
        Rect r;
        int l,i,d;
        char t[4];
        char m[128];
        Handle res, icon, GetIndResource(), GetIcon();
        long atol();

        r.top = 75;
        r.left = 20;
        r.bottom = 300;
        r.right = 310;
        n = NewWindow(0L,&r,resName[a],1,rDocProc,-1L,1,0L);

        if (n != 0) {
                SetPort(n);

                l = CountResources(atol(resName[a]));   /* how many? */

                if (l > 0) {
                        sprintf(scrap,"%d Resources",l);
                        MoveTo(10,12);
                        DrawString(scrap);      /* say it */
                        for(i=1;i<=l;i++) {
                                res = GetIndResource(atol(resName[a]),i);
                                GetResInfo(res,&d,&t,m);
                                if (strcmp(resName[a],"ICON")==0) {
                                        icon = GetIcon(d);
                                        r.top = 20 + (((i-1)/5) * 60);
                                        r.left = 10 + (((i-1)%5) * 40);
                                        r.bottom = r.top + 32;
                                        r.right = r.left + 32;
                                        if (res == 0) {
                                                FrameRect(&r);
                                        }
                                        else {
                                                PlotIcon(&r,icon);
                                        }
                                        sprintf(scrap,"#%-5d",d);
                                        MoveTo(r.left,r.bottom+12);
                                        TextSize(9);
                                        DrawString(scrap);
                                        TextSize(12);
                                }
                                else {
```

be hard to read on the screen. As the *menu_item* value is a simple *int*, it's fairly simple to put the inner *case* statements in other functions. This will help to reduce the overall real estate of the main menu switch without really changing the logic of the program.

One of the most pleasing aspects of this program is its use of the *SFGetFile( )* function. The pleasing bit is that with fewer than half a dozen lines of code you can generate a whole get file box... the thing you see if you tell MacWrite to open a file, for example.

The *SFGetFile( )* function requires a number of arguments which, like the *NewWindow( )* function we looked at a while ago, aren't all that important. The form of it is

SFGetFile(&point,"A string",0L,&longint,0L,&file_struct);

The point is a struct of the type *Point* which determines where the upper left hand corner of the box will appear on the screen. The size of the box is fixed, so there's no need to specify a rectangle. The *file_struct* is a struct of the type *SFReply*. I declared it as being global in the program as some of its information can be useful to other functions. It could have been local in this case.

There are a number of immediately useful elements of the struct. The string *file_struct.SFfName* is an ASCII file name corresponding to the name that was selected from the scrolling window in the box. The value of *file_struct.SFgood* tells us whether or not a name has been selected. If it's false, the name that's been passed should be considered to be flotsam... it usually implies that the cancel button was hit.

Note that this box has nothing to do with actually opening any files. All it does is to return the name of a selected file. Obviously, it does read all the file names from the disk to produce the scroll box. It also has a mechanism to filter only certain types of files into the scroll box... we'll check this out in another article.

More Types, More Typos

The *ErrorAlert( )* function in this program is a very simple dialog box. I haven't actually used a Macintosh alert in this thing... confronted with all the gadgets in the Mac toolbox I'm usually moved to create very elaborate programs, but this one had to be of a manageable immensity. We'll look at real alerts another time.

Dialogs are a bit of a penguin's shoe horn under C... they represent one of the few irreconcilable differences between C and Pascal as they occur on the Mac. The data structure for dialogs isn't defined in any of the header files of the compiler because the little monster changes with the number and nature of the items in the dialog.

A dialog consists of a box with some items in it. An item can be a straight up

```
                                        MoveTo(5,(12*i)+12);
                                        sprintf(scrap,"ID #%-8d %4s %s",d,t,m);
                                        DrawString(scrap);
                                }
                        }
                        WaitForRat();
                        WaitForNoRat();
                }
                else {
                        ErrorAlert("No Resources");
                }
        }
        DisposeWindow(n);
}

long atol(s)    /* returns long int of 4 byte string */
        char *s;
{
        return(
        *(s+3) * 0x1 +
        *(s+2) * 0x100 +
        *(s+1) * 0x10000 +
        *(s+0) * 0x1000000
        );
}

AboutBox()      /* draw the About box from the apple menu */
{
        WindowPtr n, NewWindow();
        Rect r;

        r.top = 75;
        r.left = 100;
        r.bottom = 110;
        r.right = 315;
        n = NewWindow(0L,&r,"",1,dBoxProc,-1L,1,0L);

        if (n != 0) {
                SetPort(n);
                MoveTo(5,10);
                DrawString("Generally functionless program");
                MoveTo(5,22);
                DrawString("Copyright (c) 1986 Steve Rimmer");
                MoveTo(5,34);
                DrawString("Belt the mouse");
        }
        WaitForRat();
        WaitForNoRat();
        DisposeWindow(n);
}

CloseFile()     /* close the resource file */
{
        CloseResFile(ResRef);
        DisableItem(menus[file_menu],3);
        DisableItem(menus[file_menu],2);
        EnableItem(menus[file_menu],1);
}

GrabEvents()    /* main event handler */
{
        EventRecord     myevent;
        int snuff_it, p;
        WindowPtr which_window;
        long menu_select, MenuSelect(), MenuKey();
        TEHandle thand;
        TEPtr tptr;
        GrafPtr saveport;

        snuff_it = 0;
        while (!snuff_it) {
                SystemTask();

                GetNextEvent(everyEvent,&myevent);
                switch(myevent.what) {
                        case mouseDown:
                                p = FindWindow(&myevent.where, &which_window);
                                switch(p) {
                                        case inSysWindow:
                                                SystemClick(&myevent,which_window);
                                                break;
```

character string... this is called a *statText* item... or a button... that's a *btnCtrl*... or a scroll bar or a text extra field or a radio button... the list is pretty full. The entire collection of items is passed to the *NewDialog( )* function in one killer struct, which I've called *ThingList* here. If you look at its definition at the top of the program, you'll see that there is an initial parameter to tell the dialog manager how many items there are followed by a number of chunks of data, each one of which must be uniquely named so as to make the struct accessible. The size of the struct varies with the number of items... and this is something that C doesn't conveniently provide for in its data definitions.

Note also that this is one of those occasions where we have a struct inside a struct. The *Rect* parameter is the rectangle inside the dialog window which the item will appear in. In the case of the button, for example, the dialog manager will call the QuickDraw *FrameRoundRect( )* function to draw a box of this size. It's also worth pointing out that this rectangle is specified in local co-ordinates, that is, relative to the origin of the dialog window, not to the origin of the window that the dialog opened into. As such, if we subsequently move the dialog all the items will stay where they were in the dialog box, moving around the screen with it.

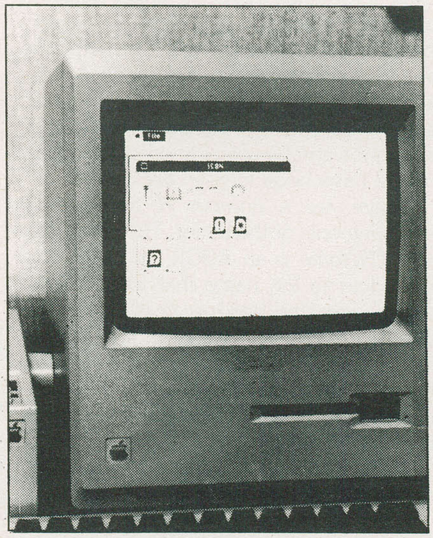
The handling of strings in the dialog structure is particularly nasty. These must be passed as Pascal strings... the compiler doesn't tidy them up for us. A C string consists of any number of bytes with a zero byte at the end. A Pascal string is a length byte followed by the string itself. As such, we have to supply the length byte to the struct.

This makes it highly nasty to change the dialog programmatically. In fact, I've done this here, but only by cheating. The dialog manager finds the start of one dialog item's data definition by working its way down through the end of the string of the last item based on the length byte. As such, the length has to be correct. I've managed to have a variable length string here only because I've made it the last item in the dialog. If I'd tacked a third item onto the end of this, the third item wouldn't have shown up and the dialog manager might have become very confused.

The *NewDialog( )* behaves very much like the *NewWindow( )*, with the exception of its being passed a handle to the item structure.

The real heart of the program is the *ShowAll( )* function, which gathers up the names of all the available resources. You can find out how many available resources there are by calling *CountTypes( )*, which returns the total as an *int*. Having done this, calling *GetIndType( )* for each value between zero and the total will return the name in a string of four bytes. This is what Pascal calls a "packed array", and it embodies some trickiness.

```
        case inMenuBar:
                menu_select = MenuSelect(&myevent.where);
                snuff_it = menu_command(menu_select);

                                        break;
                                default:
                                        break;
                        }
                break;
        }
    }
}

int     menu_command(menu_select) /* do something with the menus */
        long menu_select;
{
        int     which_menu,which_item;
        int i;
        char name[64];

        which_menu = (int)(menu_select >> 16);
        which_item = (int)(menu_select);

        i = 0;
        HiliteMenu(which_menu);
        switch(which_menu) {
        case apple_menu:
                switch(which_item) {
                        case 1:
                                AboutBox();
                                break;
                }
                break;
        case file_menu:
                switch(which_item) {
                        case 1:
                                GetFile();
                                break;
                        case 2:
                                CloseFile();
                                break;
                        case 3:
                                ShowAll();
                                break;
                        case 4:
                                i = 1;
                                break;
                }
                break;
        }
        HiliteMenu(0);
        return(i);
}

InstallMenuBar()        /* create the menu bar */
{
        MenuHandle NewMenu();

        menus[apple_menu] = NewMenu(apple_menu,"\024");
        AppendMenu(menus[apple_menu],"About Peeker");
        InsertMenu(menus[apple_menu],0);

        menus[file_menu] = NewMenu(file_menu,"File");
        AppendMenu(menus[file_menu],"Open;Close(;List(;Get Lost");
        InsertMenu(menus[file_menu],0);

        DrawMenuBar();
}

WaitForRat()            /* kills time 'til the rat gets belted  */
{
        while ( button() == 0) {
        }
}

WaitForNoRat()          /* kills time 'til the rat gets released */
{
        while ( button() != 0) {
        }
}
```



The string which gets returned isn't really a C string... it has no terminating null... or a Pascal string, as it has no length byte. We can print it easily using *printf*... or *sprintf* in this case... by having it printed as 4s, that is, a string of no more than four bytes. The real tricky bit is how it's passed to other Mac functions.

The resource names are stored in the global array *resName* as strings.

The rest of the *ShowAll( )* function is a simple custom menu. Having printed all the resource names in the window, this last half of the function will wait for a mouse click. If you click on one of the names it will invert the name and call *SeeResource( )* with the number of the resource you've selected.

The *SeeResource( )* function is the most complicated bit of code in the program. It gets handles to each of the resources of the type passed to it and displays the information they represent. This is where the use of the resource names gets really freaky. If normal humans were to see the string "ICON", for example, we'd look at it and say that it's a four byte string. It's not... at least as far as the Mac is concerned. It's a long integer. It's actually four bytes arrayed in a way that looks like a string and can be printed, but it's interpreted as a thirty-two bit number. This sounds funky, but it does yield a unique number for each unique name.

The function *atol( )* here converts a four byte string into a long *int*... if you look at it you'll probably have a better idea as to how the numbers are stored as resource names.

In all the cases but one, the *SeeResource( )* function just gets the usual information about the resources, that is, their numbers, names and types. The names are optional, and are often omitted. You find them, however, on fonts, for example. These resources are, however, actual data in many cases, which can be represented in the way it's really used. To show how this happens, I've had the program trap the ICON resource and, rather than just printing some fairly meaningless strings, it actually plots all the icons it can find.

The handle returned by *GetIndResource( )* holds the data for an actual icon in this resource, and can be passed to *PlotIcon( )* along with a rectangle to plot the icon in. This actually uses the Mac toolbox *CopyBits( )* function... something else we'll talk about at a later time... and, as such, it will do scaling for you. If you don't make the rectangle thirty-two by thirty-two... the usual size of icons... it will scale the icon to fit what you do give it. Thus can look highly strange.

Slack Macs

The joy of programming the Mac... especially if you're really doing it for your own amusement... is that your programs never really need be finished. There are so many effects and things to play with in the Macintosh operating system that you can always add something to your applications, or get something a bit stranger with a few more lines and another compile.

It's also very satisfying to create applications that do something un-Mac like. One can have weird menus with icons in 'em, bizarre dialogs or all sorts of things that don't ever appear in respectable Mac applications. The temptation is awesome, and the potential for doing so is no further away than your drives.

What a cosmic slice... **CN!**

Circle No. 11 on Reader Service Card

Serial In C

In pushing at the limits of C, one quickly encounters the IBM PC's serial ports, something that C doesn't like to deal with. Here's a troll though the random fields for some solutions.

by Steve Rimmer

There are a number of things that C is extremely good at. It handles data well, for example, and it's a hoot for doing things with files. However, by virtue of its universality, even really powerful C compilers, such as the Lattice version three package the attendant programs in this article were written in, usually lack some of the more essential bits of hardware support for the computers they run on.

Serial communications is one of those things that is unpleasantly difficult to get together in C on the IBM PC. There are two fairly predictable ways to approach it... we'll talk about them briefly in a moment... both of which are notable in that they don't work. The PC has brilliant hardware to support serial communications and appallingly bad firmware to deal with it.

In this feature we're going to look at a way to deal with the serial ports of the PC from C that *does* work. We're also going to check out a simple XMODEM style file transfer routine that will allow you to move files to and from the IBM with relative security from gorches.

Oh C Can You Say...

The principles of serial communication at a programming level are pretty simple. You wait for a status flag, and get a byte from a port when it becomes set. Likewise, sending data out involves checking another status flag, and outputting a byte when it's cool. One should be able to handle all this from within a C program with nothing more than the *inp( )* and *outp( )* functions.

Unfortunately, there are a number of hassles in this. The primary one is that the PC isn't really fast enough to handle high speed communications like this... anything over about twelve hundred baud tends to have it lose characters periodically.

You might also have noticed that there is a PC BIOS function that handles serial communications. This can be accessed by the *int86( )* function under Lattice C... and it, too, looks promising. However, it doesn't do a great deal more than watching the flags... it's largely in there to take care of serial printers, rather than actual, civilized telecommunications.

There are no C functions that do sensible communications with the serial ports. This is a colossal drag, of course, and very barbaric. As such, we're going to write some.

Listing One

```
COMMENT -

        INTERRUPT DRIVEN SERIAL I/O
        MACHINE LANGUAGE MODULE
        FOR C (PROBABLY WORKS WITH
        ANY DECENT PC C COMPILER)

        copyright (c) 1986 steve rimmer

        -

FALSE           EQU     0
TRUE            EQU     NOT FALSE

LONG_           EQU     FALSE           ;TRUE FOR LARGE MEMORY MODEL
COM1            EQU     TRUE            ;SET ONE TRUE FOR YOUR SERIAL
COM2            EQU     FALSE           ;PORT ASSIGNMENT

                IF      LONG_
A_OFF           EQU     6
                ELSE
A_OFF           EQU     4
                ENDIF


DGROUP          GROUP   DATA
DATA            SEGMENT WORD PUBLIC 'DATA'
                ASSUME  DS:DGROUP

HEAD_BUFF       DW      ?       ;PLACE TO STORE HEAD OF BUFFER
TAIL_BUFF       DW      ?       ;PLACE TO STORE TAIL OF BUFFER
SERIAL_BUFFER   DW      ?       ;POINTER TO BUFFER
BUFFER_LENGTH   DW      ?       ;LENGTH OF BUFFER

                PUBLIC  IO_VERSION
IO_VERSION      DB      'Serio V1.0 copyright (c) 1986 steve rimmer '
                IF      COM1
                DB      'IBM PC COM1:'
                ENDIF
                IF      COM2
                DB      'IBM PC COM2:'
                ENDIF
                DB      0

DATA            ENDS

PGROUP          GROUP   PROG
PROG            SEGMENT BYTE PUBLIC 'PROG'
                ASSUME  CS:PGROUP

;THIS ROUTINE SETS THE BAUD RATE TO ARG1
;
;       CALLED AS
;       baud_rate(baud)
;       unsigned baud;
;

                PUBLIC  BAUD_RATE
                IF      LONG_
BAUD_RATE       PROC    FAR
                ELSE
BAUD_RATE       PROC    NEAR
                ENDIF

                PUSH    BP
                MOV     BP,SP

                IF      COM1
                MOV     DX,03FBH
                ENDIF
                IF      COM2
                MOV     DX,02FBH
                ENDIF

                MOV     AL,80H
                OUT     DX,AL           ;SET DLAB

                MOV     AX,[BP+A_OFF]   ;GET BAUD RATE

                IF      COM1
                MOV     DX,03F8H
                ENDIF
                IF      COM2
                MOV     DX,02F8H
                ENDIF

                OUT     DX,AL           ;SET IT

                IF      COM1
                MOV     DX,03F9H
                ENDIF
                IF      COM2
                MOV     DX,02F9H
                ENDIF

                MOV     AL,AH
                OUT     DX,AL           ;SET LSB

                IF      COM1
                MOV     DX,03FBH
                ENDIF
                IF      COM2
                MOV     DX,02FBH
                ENDIF

                MOV     AL,07H          ;SET CFW
                OUT     DX,AL

                POP     BP
                RET
BAUD_RATE       ENDP


;THIS ROUTINE CLEARS THE BUFFER AND RESETS THE POINTER
;
;               CALLED AS
;               reset_serial();
;

                PUBLIC  RESET_SERIAL

                IF      LONG_
RESET_SERIAL    PROC    FAR
                ELSE
RESET_SERIAL    PROC    NEAR
                ENDIF

                MOV     AX,[SERIAL_BUFFER]
                MOV     [HEAD_BUFF],AX
                MOV     [TAIL_BUFF],AX  ;INITIALIZE BUFFER POINTERS
                RET

RESET_SERIAL    ENDP

;THIS ROUTINE INSTALLS THE VECTOR, SETS UP BUFFER TO &ARG1, LEN ARG2
;
;               CALLED AS
;               set_serial(buffer,length);
;               *char buffer;
;               int length;
;

                PUBLIC  SET_SERIAL

                IF      LONG_
SET_SERIAL      PROC    FAR
                ELSE
SET_SERIAL      PROC    NEAR
                ENDIF


                PUSH    BP
                MOV     BP,SP
                MOV     DX,[BP+A_OFF]                   ;GET BUFFER ADDRESS
                MOV     AX,[BP+A_OFF+2]                 ;GET BUFFER LENGTH

                MOV     [SERIAL_BUFFER],DX
                MOV     [BUFFER_LENGTH],AX

                MOV     AX,[SERIAL_BUFFER]
                MOV     [HEAD_BUFF],AX
                MOV     [TAIL_BUFF],AX                  ;INITIALIZE BUFFER POINTERS

                CLI                                     ;TURN OFF INTERUPTS
                PUSH    DS

                MOV     DX,OFFSET HANDLER
                PUSH    CS
                POP     DS
                MOV     AH,25H                          ;SET VECTOR

                IF      COM1
                MOV     AL,0CH                          ;SET UP VECTOR FOR IRQ
                ENDIF
                IF      COM2
                MOV     AL,0BH
                ENDIF

                INT     21H
                POP     DS

                IN      AL,21H
                IF      COM1
                AND     AL,11101111B                    ;MASK FOR INTERUPT BIT
                ENDIF
                IF      COM2
                AND     AL,11110111B
                ENDIF

                OUT     21H,AL

                IF      COM1
                MOV     DX,03FBH                        ;LINE CONTROL FOR 8250...
                ENDIF
                IF      COM2
                MOV     DX,02FBH
                ENDIF

                IN      AL,DX
                AND     AL,7FH
                OUT     DX,AL

                IF      COM1
                MOV     DX,03F9H                        ;INTERUPT ENABLE FOR 8250...
                ENDIF
                IF      COM2
                MOV     DX,02F9H
                ENDIF

                MOV     AL,01H
                OUT     DX,AL

                IF      COM1
                MOV     DX,03FCH                        ;MODEM CONTROL FOR 8250...
                ENDIF
                IF      COM2
                MOV     DX,02FCH
                ENDIF
                MOV     AL,08H
                OUT     DX,AL
```

```
                STI
                POP     BP
                RET
SET_SERIAL      ENDP

;THIS ROUTINE SENDS A CHARACTER OUT TO THE SERIAL PORT
;
;               CALLED AS
;               put_serial(c);
;               int c;
;               returns zero if time out

                PUBLIC  PUT_SERIAL

                IF      LONG_
PUT_SERIAL      PROC    FAR
                ELSE
PUT_SERIAL      PROC    NEAR
                ENDIF

                PUSH    BP
                MOV     BP,SP

                IF      COM1
                MOV     DX,03FDH
                ENDIF
                IF      COM2
                MOV     DX,02FDH
                ENDIF

                SUB     CX,CX
PUT_SERIAL1:    IN      AL,DX           ;GET MODEM STATUS
                AND     AL,20H          ;CHECK CTS AND DSR
                CMP     AL,20H
                JE      PUT_SERIAL2
                LOOP    PUT_SERIAL1
PUT_SERIAL2:
                MOV     AX,[BP+A_OFF]   ;GET BYTE TO SEND

                IF      COM1
                MOV     DX,03F8H
                ENDIF
                IF      COM2
                MOV     DX,02F8H
                ENDIF

                OUT     DX,AL           ;SEND IT
                MOV     AX,CX
                POP     BP
                RET
PUT_SERIAL      ENDP

;THIS ROUTINE RETURNS A CHARACTER FROM THE BUFFER, OR FALSE IF NONE
;
;               CALLED AS
;               c = get_serial();
;               int c;
;
                PUBLIC  GET_SERIAL

                IF      LONG_
GET_SERIAL      PROC    FAR
                ELSE
GET_SERIAL      PROC    NEAR
                ENDIF

                MOV     BX,[HEAD_BUFF]
                CMP     BX,[TAIL_BUFF]  ;SEE IF HEAD = TAIL...
                JNE     GET_BUFFER1     ;... NO BYTE IF IT DOES
                SUB     AX,AX           ;RETURN A NULL
                RET

GET_BUFFER1:    SUB     AX,AX
                MOV     AL,[BX]         ;GET THE BYTE
                CALL    BUMP_BUFF       ;INCREMENT THE POINTER
                MOV     [HEAD_BUFF],BX  ;AND SAVE IT
                RET
GET_SERIAL      ENDP

;THIS ROUTINE TESTS FOR WAITING DATA, RETURNS 0 IF NOTHING
;
;               CALLED AS
;               c = test_serial();
;               int c;
;

                PUBLIC  TEST_SERIAL

                IF      LONG_
TEST_SERIAL     PROC    FAR
                ELSE
TEST_SERIAL     PROC    NEAR
                ENDIF

                PUSH    BP
                MOV     AX,[HEAD_BUFF]  ;GET BUFFER HEAD POINTER
                SUB     AX,[TAIL_BUFF]
                POP     BP
                RET
TEST_SERIAL     ENDP

;THIS ROUTINE UNDOES THE VECTOR
;
;               CALLED AS
;               c = kill_serial();
;

                PUBLIC  KILL_SERIAL

                IF      LONG_
KILL_SERIAL     PROC    FAR
                ELSE
KILL_SERIAL     PROC    NEAR
                ENDIF

                CLI
                IN      AL,21H

                IF      COM1
                OR      AL,00010000B            ;UNMASK BIT FOR INTERUPT ENABLE
                ENDIF
                IF      COM2
                OR      AL,00001000B
                ENDIF

                OUT     21H,AL

                IF      COM1
                MOV     DX,03FBH                ;LINE CONTROL FOR 8250
                ENDIF
                IF      COM2
                MOV     DX,02FBH
                ENDIF
                IN      AL,DX
                AND     AL,7FH
                OUT     DX,AL

                IF      COM1
                MOV     DX,03F9H                ;INTERUPT ENABLE FOR 8250...
                ENDIF
                IF      COM2
                MOV     DX,02F9H
                ENDIF
                MOV     AL,0
                OUT     DX,AL

                IF      COM1
                MOV     DX,03FCH                ;MODEM CONTROL FOR 8250...
                ENDIF
                IF      COM2
                MOV     DX,02FCH
                ENDIF
                MOV     AL,0
                OUT     DX,AL

                STI
                RET
KILL_SERIAL     ENDP

;THIS ROUTINE HANDLES THE SERIAL INTERUPT
HANDLER:        STI                     ;ENABLE OTHER INTERUPTS
                PUSH    AX
                PUSH    BX
                PUSH    DX
                PUSH    SI
                PUSH    DS

                MOV     AX,DATA
                MOV     DS,AX

                IF      COM1
                MOV     DX,03F8H                ;RECEIVE BUFFER FOR 8250...
                ENDIF
                IF      COM2
                MOV     DX,02F8H
                ENDIF

                IN      AL,DX
                MOV     BX,[TAIL_BUFF]          ;GET POINTER TO TAIL
                MOV     SI,BX
                CALL    BUMP_BUFF               ;INCREMENT THE POINTER
                MOV     [TAIL_BUFF],BX          ;AND SAVE IT
                MOV     [SI],AL

                CLI
                MOV     AL,20H                  ;SIGNAL END OF INTERUPT
                OUT     20H,AL
                STI

                POP     DS
                POP     SI
                POP     DX
                POP     BX
                POP     AX
                IRET                            ;RETURN TO CALLING CODE

;THIS ROUTINE INCREMENTS A BUFFER POINTER IN BX
BUMP_BUFF:      PUSH    AX
                MOV     AX,[SERIAL_BUFFER]
                ADD     AX,[BUFFER_LENGTH]
                INC     BX

                CMP     BX,AX
                JGE     BUMP_BUFF1
                POP     AX
                RET

BUMP_BUFF1:     MOV     BX,[SERIAL_BUFFER]
                POP     AX
                RET

PROG            ENDS
                END
```

Serial In C

In order to get decent, reliable serial communications happening on a PC one needs set up an interrupt driven serial port handler. We had a look at a terminal program based on this principle in the April 1986 edition of Computing Now!. The code here is similar, although we're going to have to adapt it to the dictates of C.

It would be difficult... oh, probably impossible... to write the actual interrupt mechanism in C, and a bit inconvenient to do the low level handlers in it. It's much more reasonable to write this stuff in assembler and link the resulting module to one's C programs at compile time. If a suitable object module is kicking around your compiler disk somewhere and you've got its name in your linking command, you can treat these home made functions as if they were part of the Lattice library.

It's relatively infrequent that one needs write assembly language modules for Lattice C... it's an extremely rich language, one that doesn't need a lot of propping up. Unless you've gotten into something freaky like this project, you may well never have had the cosmic joy of getting something like this together. It isn't as daunting as it looks.

I've constructed this bit of assembler code... the one in the first listing... a bit differently than they want you to do it in the Lattice manual. Lattice avails one of a number of macros and other useful bits that don't live in other compilers... I've written this code out without the macros so it should be applicable to virtually any PC C package that links with the Microsoft linker.

In order to write assembly language functions for C, it's necessary to understand how C passes parameters between its functions, as the compiler doesn't differentiate between home made functions and ones that came with it or were written in C. It will expect this weird assembly code to behave as would authentic C routines.

You can pass any number of arguments *to* a C function, but you can only have it directly return one. In the case of these functions, we're going to restrict ourselves to passing a few basic data types, mostly *ints* and one pointer. Pointers are essentially passed as *ints* here... C programmers will know this to be despicable, of course, but the assembly language side of things doesn't care. Numbers and pointers... at least, simple pointers like we'll be using... are the same under assembly language.

There are a number of things that... for the sake of convenience and keeping the arguments to the functions down to a manageable number... we must decide about the assembly language module at assembly time. For example, we'll choose the size of the compiler model. The small models, the ones with sixty-four kilobyte code limits, are "short"... they can use near calls. If you are going to be linking the module into a small model program, set the LONG_ equate false.

You'll also have to decide which COM port you're going to use, setting one of the appropriate equates true.

The IO_VERSION label points to a string which will allow your programs to identify the module once it has been linked into your code. This is probably a worthwhile thing to use. If you declare *io_version* as *extern* under a C program, doing

puts(&io_version);

will print the string.

This is not all that heavy, I know.

The BAUD_RATE function is a good example of how one passes parameters to an assembly language function. C gets this together by pushing the arguments onto the stack, calling the function and then pulling the arguments off. This sounds really funky, but it allows virtually unlimited nesting of functions and an unlimited number of arguments. It does have some drawbacks, though... machine language functions have

Listing Two

```
/*
        basic XMODEM send/receive program
        possibly not the most elegant implimentation
        of the protocol, but it works and it's short.
        copyright (c) 1986 steve rimmer
        Western civilization is still
        waiting for its initial NAK.
*/

#include "stdio.h"
#include "stdlib.h"
#include "fcntl.h"
#include "dos.h"
#define error_ok        0
#define error_timeout   1
#define error_eot       2
#define error_bnum      3
#define error_checksum  4
#define error_gotnak    5
#define error_maxerr    6
#define error_can       7
#define error_nonak     8
#define error_filex     9
#define error_makef     10
#define error_noacc     11
#define error_nofile    12
#define error_nosec     13
#define error_eof       14
#define error_nak       15
#define ser_timeout     10      /* number of seconds to timeout */
#define max_sererr      10      /* maximum number of errors */
#define SOH             1
#define ACK             6
#define NAK             21
#define CAN             24
#define EOT             4
static char error_mess[16] [20] = {
        { "ok"                },
        { "timeout"           },
        { "received ok"       },
        { "bad block number"  },
        { "bad checksum"      },
        { "NAK recieved"      },
        { "too many errors"   },
        { "aborted"           },
        { "no SOH received"   },
        { "file exists"       },
        { "can't open file"   },
        { "access denied"     },
        { "file not found"    },
        { "timeout on sector" },
        { "end of file"       },
        { "got NAK"           }
};
        int ser_error;                  /* the current error code */

        char buffer[134];               /* buffer for sector */
        unsigned l_block,h_block;       /* block numbers */
        FILE *xmod_fp;                  /* pointer to file */

main(argc,argv)
        int argc;
        char *argv[];
{
        char *p,*malloc();              /* pointer to serial buffer */

        if (argc > 1) {                 /* make sure we have arguments */
                l_block = 1;                    /* initialize block numbers */
                h_block = 0xfe;
                if((p = malloc(800)) != 0) {
                        set_serial(p,800);      /* set 800 byte buffer */
                        baud_rate(0x000c);      /* set baud rate to 9600 */
                        puts("\non line ");
                        if(tolower(*argv[1]) == 'r') {  /* check for receive */
                                xmodem_rec(argv[2]);
                        }
                        else if(tolower(*argv[1]) == 's') {     /* try send */
                                xmodem_send(argv[2]);
                        }
                        else {
                                puts("\nbad option");
                        }
                        puts("\nback to dos");
                        kill_serial();          /* unhook serial drivers */
                        free(p);                /* free serial buffer memory */
                }
                else {
                        puts("\nmemory gorch");
                }
        }
        else {
                puts("\gotta have an argument");
        }
}

xmodem_rec(s)   /* receive file s xmodem */
        char *s;
{
        xmake_file(s);                          /* create file */
        if(ser_error == error_ok) {
                get_file(s);                    /* get the contents */
                fclose(xmod_fp);                /* close it */
        }
        else {
                say_error();
        }
}

xmodem_send(s)  /* send file s xmodem */
        char *s;
{
        struct FILEINFO f;
        int c;

        xopen_file(s);                          /* open file */
        if(ser_error == error_ok && dfind(&f,s,0) == 0) {
                c = f.size / 128L;              /* get sector count */
                printf("\n%d sectors to send",c);
                put_file(c);                    /* send file */
                fclose(xmod_fp);                /* and close it */
        }
        else {
                say_error();
        }
}

xopen_file(s)   /* open file for send */
        char *s;
{
        if(access(s,0) == 0) {                  /* check for file */
                if((xmod_fp = fopen(s,"rb")) != NULL) {
                        ser_error = error_ok;
                }
                else {
                        ser_error = error_noacc;
                }
        }
        else {
                ser_error = error_nofile;
        }
}

xmake_file(s)   /* create file for recieve */
        char *s;
{
        if(access(s,0) != 0) {                  /* check if file exists */
                if((xmod_fp = fopen(s,"wb")) != NULL) {
                        ser_error = error_ok;
                }
                else {
                        ser_error = error_makef;
                }
        }
        else {
                ser_error = error_filex;
        }
}

put_file(s)     /* send a file */
        int s;
{
        int e = 0;
        int c = 1;

        reset_serial();                 /* clear buffer */
        ser_error = -1;                 /* get initial NAK */
        while (e < max_sererr && ser_error != error_ok) {
                printf("\nawaiting initial NAK %d",c);
                get_xserial();
                ++c;
        }
        if (e < max_sererr && ser_error == error_ok) {
                c = 1;
                do {
                        printf("\nsending sector %d",c);
                        put_sector();   /* send a sector */
                        ++c;
                        bump_block();   /* adjust block numbers */
                } while (ser_error == error_ok && c <= s);

                if (c >= s ) {
                        put_serial(EOT);        /* send end of transmission */
                        puts("\nall transfers complete");
                }
        }
        else {
                ser_error = error_nosec;
        }
}

put_sector()    /* send one sector */
{
        int i;
        short e = 0;
        unsigned a;

        *buffer = SOH;
        *(buffer + 1) = l_block & 0xff;
        *(buffer + 2) = h_block & 0xff;
        i = 3;
        while( i < 131) {               /* get one sector into buffer */
                *(buffer+i) = fgetc(xmod_fp);
                ++i;
        }
        i = 0;
        a = 0;
        while(i < 131 ) {               /* calculate checksum */
                a = a + *(buffer+i) & 0xff;
                ++i;
        }
        *(buffer+131) = a & 0xff;       /* place checksum */
        do {
                i = 0;                  /* send sector */
                while ( i < 132) {
                        put_serial(*(buffer+i));
                        ++i;
```

```
		}
		i = get_xserial();	/* get back result */
		if (ser_error == error_ok) {
			switch (i) {
				case ACK:
					ser_error = error_ok;
					break;
				case NAK:
					ser_error = error_nak;
					say_error();
					break;
				case CAN:
					ser_error = error_can;
					say_error();
					break;
				default:
					printf("\ngot chr$(%d)",i);
					ser_error = error_nak;
					break;
				}
			}
			else {
				say_error();
				ser_error = error_nak;
			}
		++e;	/* loop 'til ok or bad error */
	} while (ser_error == error_nak && e < max_sererr);
}

get_file()	/* get a file */
{
	int e = 0;
	int c = 1;
	int i;

	reset_serial();		/* clear buffer */
	while (e < max_sererr && test_serial() == 0) {
		printf("\nsending NAK %d",e);
		put_serial(NAK);
		sleep(5);
		++e;		/* send initial NAK */
	}
	if (e < max_sererr) {
		e = 0;
		do {
			printf("\nawaiting sector %d",c);
			i = get_sector();	/* get one sector */
			switch(i) {		/* check results */
			case EOT:
				ser_error = error_eot;
				say_error();
				put_serial(ACK);
				break;
			case CAN:
				ser_error = error_can;
				say_error();
				break;
			case SOH:
				if (ser_error == error_ok) {
					if(check_sum() == 0) {
					if (blocknum() == 0) {
						++c;
						write_sector();
						e = 0;
						reset_serial();
						put_serial(ACK);
					}
					else {
						ser_error = error_bnum;
						++e;
						say_error();
						sleep(2);
						reset_serial();
						put_serial(NAK);
					}
				}
				else {
					ser_error = error_checksum;
					++e;
					say_error();
					sleep(2);
					reset_serial();
					put_serial(NAK);
				}
			}
			else {
				say_error();
				++e;
				put_serial(NAK);
			}
			break;
		default:
			printf("\ngot chr$(%d) instead of SOH",i);
			break;
		}

	} while (ser_error != error_eot &&
		ser_error != error_can &&
		e < max_sererr);
	}
	else {
		ser_error = error_nonak;
		say_error();
	}
}

blocknum()	/* check the block numbers */
{
	unsigned a,b;

	a = *(buffer+1) & 255;
	b = *(buffer+2) & 255;
	if (l_block == a && h_block == b) {
		bump_block();
		return(0);
	}
	else {
		return(1);
	}
}

bump_block()	/* bump up the block numbers */
{
	++l_block;
	l_block = l_block & 0xff;
	--h_block;
	h_block = h_block & 0xff;
}

check_sum()	/* calculate checksum for sector */
{
	int i;
	unsigned a,b;

	a = 0;
	i = 0;
	while(i < 131 ) {
		a = a + *(buffer+i) & 0xff;
		++i;
	}
	b = *(buffer + 131) & 0xff;
	if (a == b) {
		return(0);
	}
	else {
		return(1);
	}
}

write_sector()	/* write the current sector to the file */
{
	int i;

	i = 3;
	while (i < 131) {
		fputc(*(buffer+i),xmod_fp);
		++i;
	}
}

say_error()	/* show the error message */
{
	printf("\n%s",error_mess[ser_error]);
}

get_sector() /* get a sector into console */
{
	int i,c;

	ser_error = error_ok;
	if ((c = get_xserial()) == SOH && ser_error == error_ok) {
		i=1;
		*buffer = SOH;
		while (i < 132 && ser_error == error_ok) {
			*(buffer+i) = get_xserial();
			++i;
		}
		return(SOH);
	}
	else {
		sleep(5);
		return(c);
	}
}

get_xserial()	/* return serial byte or set ser_error */
{
	long s,p;
	int c;

	time(&s);
	time(&p);
	c = 0;
	while(p < s + ser_timeout && (c = test_serial()) == 0) {
		time(&p);
	}
	if (c != 0) {
		ser_error = error_ok;
		c = get_serial();
	}
	else {
		ser_error = error_timeout;
	}
	return(c);
}

sleep(n)	/* sleep for n seconds */
	int n;
{
	long t,p;
	int d;

	time(&p);
	d = 0L;
	while(d < n) {
		time(&t);
		d = t - p;
	}
}
```

to know how many arguments to expect, and they'll invariably inhale garbage if they're given the wrong number.

The baud rate function is given a single argument. It you look at the listing, you'll see how it's extracted from the stack. Note that there's no need to do any sort of freaky adjusting for the stack when we pull an argument from it... we aren't actually popping anything off it, but merely copying data from behind the stack pointer.

The baud rate function, by the way, could have been handled in C quite nicely. However, it takes less code this way. Its function is to output the divisors necessary to set the baud rate of the serial port in question. The integer that's passed to it is actually the two bytes of the baud rate divisor. In the program in listing two I've used it to set the baud rate to ninety-six hundred baud. If you write a more sophisticated program you might want to make this variable... here are the baud rate divisors for a number of common baud rates.

```
#define   baud300     0x0180
#define   baud1200    0x0060
#define   baud2400    0x0030
#define   baud9600    0x000c
#define   baud19200   0x0006
```

Returning values from an assembly language function... assuming we want to return something simple, like an *int*, is really straight up. One simply stashes the value in the AX register. If the calling function is looking for a returned value it will grab the contents of AX. Otherwise, it will ignore them.

Constant Interruptions

The actual process of making this serial driver work is a bit more complex than it looks. The SET_SERIAL function tells part of the tale. When it's run, it must be passed a pointer to a buffer and the length of the buffer. The buffer will get used to handle serial data as it comes in through the ports.

Under normal conditions, when a serial byte shows up at one of the PC's ports, the computer sits passively by, waiting for something to come and read it out. If a second byte shows up before the first one has been read, the first one gets trashed. However, it's possible to program the PC to be a lot more decisive about these things. The setup code here tells it to throw an interrupt every time a character is ready to be read out of its serial port.

Because the PC has piles of interrupts available to it, there are individual ones that are devoted to nothing but the serial ports, these being 0CH for the primary port and 0BH for the secondary one. Having programmed one of the serial ports to pull an interrupt when it has data to be picked up, the hardware equivalent of an INT instruction will happen every time a character is ready for reception.

Normally, the vectors for these interrupts are undefined. If one were to get thrown with the computer having been newly powered up, chances are it would leap off into never never land and hang. As such, aside from merely enabling the appropriate serial interrupt, we also have to provide it with something to do should it get thrown. The "something to do" is the code HANDLER.

The handler is never called by a program, but only by the interrupt mechanism of the PC. If it's called, it saves all the registers it might corrupt on the stack and grabs the waiting serial byte. It saves serial bytes in a large buffer... the one we passed the setup program from C... and then returns to the place where the computer was when the interrupt happened. Aside from gobbling a bit of processor time, then, the handler is invisible to the program that is running on the system when it's called.

The GET_SERIAL routine reads characters out of the buffer... if any are there... and returns them as C compatible values. The TEST_SERIAL function returns the number of bytes in the buffer... if it returns zero, there's nothing to read out.

The last important function is KILL_SERIAL. It disables the interrupt mechanism before the program returns to DOS. It's essential to do this, as, if we don't, serial bytes which wander in after the program has terminated will activate the interrupt function and have the computer leap to what it thinks is a handler... with unpredictable results.

There's a more complete explanation of this code in the April 1986 Computing Now!. However, you don't have to understand all its complexities to use it... it behaves like a black box as far a C is concerned.

Free The Modem Seven

The program in listing two is a moderately useful bit of code if you are faced with the prospect of having to move files between computers. It will send and receive files to any other terminal package which can generate the XMODEM protocol. In order to have it send files the other system must be set up to deal with checksums, as it isn't into CRCs.

You'd call the program as

XMODEM S WOMBAT.DOC

to send a file from your computer, or

XMODEM R WOMBAT.DOC

to receive a file coming at you from another system.

There's a pretty tight explanation of the workings of the XMODEM protocol in the Ultraboot article in the September 1985 edition of Computing Now!... if some of this sounds a bit mysterious, you might want to refer to that piece for some clarification.

Data under XMODEM is sent in sectors of a hundred and twenty-eight bytes. Actually, the sectors that are sent are a rather longer than this. Prior to the actual data there is a byte called an SOH, or start of header, to tell the computer at the other end that something juicy is coming. The next two bytes are a block number from zero to two hundred and fifty-five and the twos complement of it. Since the start of header character is actually character one, this plus the two block number characters will add up to a byte of zero if they aren't corrupted in transmission. This is important.

The next bytes are the data itself. This is followed by a checksum byte, which is the sum of the three header bytes and the data. The receiving computer will check that the block numbers correspond to the ones it expects to get and that the checksum it has received matches the one it calculates itself based on the received sector.

If the sector looks good the receiving computer will send an ACK character. If it's a dud, it will send a NAK. If the sending computer gets a NAK it will resend the same sector. The receiving computer will only save the sector to disk if it has passed muster.

When the whole file has been sent the sending computer will sent an EOT character, and the receiving computer will close the file.

The only other important bits are that the receiving computer must send a NAK to start the first sector coming at it... hence the message "waiting for initial NAK" in the program... and that either system can abort the process by sending a CAN character instead of an SOH.

You can pretty well see the process in the code in listing two. The various functions communicate between themselves through the *ser_error* variable, which serves as a flag for the numerous conditions the process can get itself into. There are a lot of potential errors, such as checksums not matching, timing out while waiting for an ACK and so on.

Hard Copy

I've kept the XMODEM program as short as possible... it's useful as it stands, but not too pretty. Once you get it going, you can make quite a lot more of it. C is a good language to build things up in.

Despite the apparent complexity of the protocol, XMODEM is surprisingly easy to code in C. It sounds very funky when you describe it, but it has a reasonably logical flow to it, and can be handled without masses of goto's or peculiar structures. This might be compared to the assembly language version of it in the September 1985 Computing Now!... which is quite convoluted. This one also took a lot less time to write and debug.

CN!

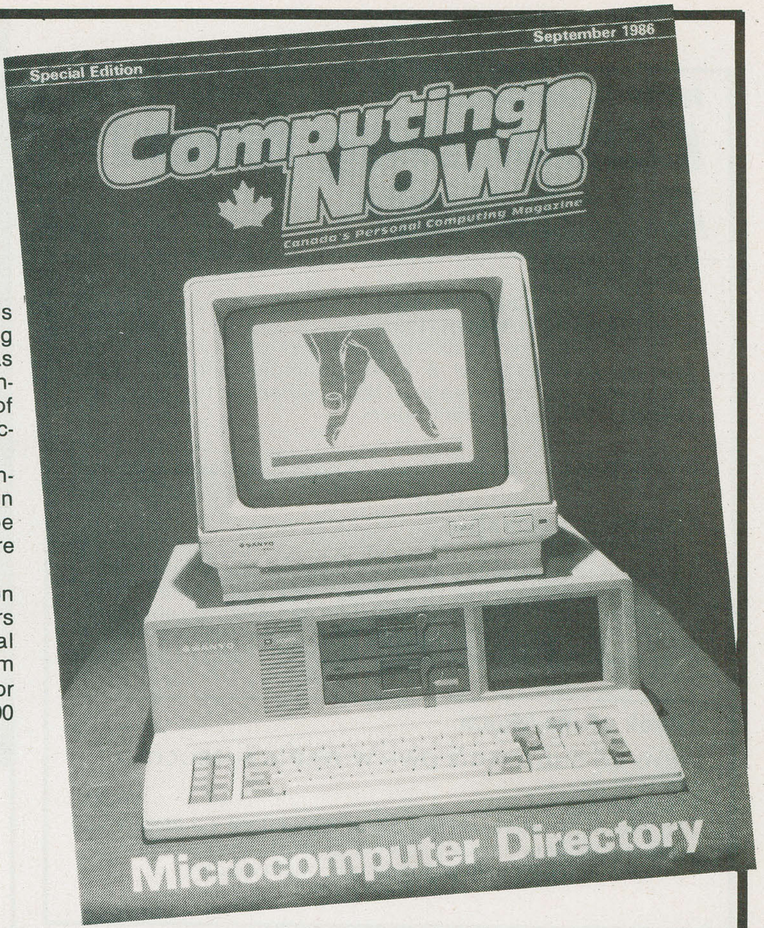
Computing Now! Microcomputer Directory

Finding the hardware or software one wants for one's microcomputer applications is very often an exercise in finding out what exists. Following this, one must locate what one has chosen, often through a labyrinth of dealers and suppliers. Conventional magazine directories and surveys can help with all of this... somewhat... but to date a comprehensive useable directory of what's available in Canada has yet to be published.

Most directories are constructed so as to be easy to compile. This one has been designed to be easy to use. Rather than selecting arbitrary categories, the items in the directory will be classified by system and within each system, in a tree structure dividing them into subclassifications.

This will result in a large directory with a lot of duplication in it. However, it will be a reference highly valued by its readers because of its resulting ease of use. It will be the essential reference work for anyone involved in microcomputers, from single system end users to small and mid-sized businesses. For advertising information contact Denis Kelly at (416) 445-5600 immediately.

Cover Date: September 1986
Copy Close: July 20th 1986



Computing Now! Order Form

Subscriptions:

Please complete reverse side of order form to start or renew a subscription.

Back Issues: $4.00 each plus 7% Ontario P.S.T.
Please circle issues desired.

1983	1984	1985	1986
April	Jan.	Jan.	Jan.
May	Feb.	Feb.	March
June	March	March	April
July	April	April	May
Aug.	May	May	June
Sept.	June	June	
Oct.	July	July	
Nov.	Aug.	Aug.	
Dec.	Sept.	Sept.	
	Oct.	Oct.	
	Nov.	Dec.	
	Dec.		

Binders:

Imprinted ☐ Electronics Today; ☐ Computing Now!
☐ Moorshead Publications $9.95 each plus 7% P.S.t.

SPECIAL PUBLICATIONS

ITEM	QTY	AMOUNT
Project Book No. 2 $3.95	....	$ ....
50 Top Project $4.95	....	$ ....
Your First Computer $3.95	....	$ ....
Computers in Samll Business $3.95	....	$ ....
Book of Computer Music $4.95	....	$ ....

BOOKSHELF — Order Form

Code (e.g. BP12)	Title (Short-form is O.K.)	Price
....	....	$ ....
....	....	$ ....
....	....	$ ....
....	....	$ ....
....	....	$ ....

SOFTWARE — Order Form

Title	Price
....	$ ....
....	$ ....
....	$ ....
....	$ ....
....	$ ....
....	$ ....
....	$ ....

SubTotal $ ....
Tax (Ontario Residents) $ ....
Postage $1.00 ....
Total Enclosed $ ....

Orders from the Bookshelf are tax exempt. Please add $1.00 for postage. Remember to put your name and address on reverse side. See over for mailing details. Specials and Software add 7% PST.

New & Revised
Computing Now! Bookshelf

PH107: APPLE LOGO PRIMER
G. BITTER & N. WATSON (1983) **$19.95**
A pictorial starter book that will make LOGO easy for anyone. Includes easy to follow examples and reference tables. Also included is a workshop outline for teachers and leaders who want to train others.

THE BASIC COOKBOOK.
TAB No. 1055: **$10.45**
BASIC is a surprisingly powerful language . . . if you understand it completely. This book, picks up where most manufactuers' documentation gives up. With it, any computer owner can develop programs to make the most out of his or her machine.

Book of the Month

HOW TO BUILD YOUR OWN WORKING MICROCOMPUTER
TAB No. 1200 **$16.45**
An excellent reference or how-to manual on building your own microcomputer. All aspects of hardware and software are developed as well as many practical circuits.

Product Mart
Where Buyers Find Sellers

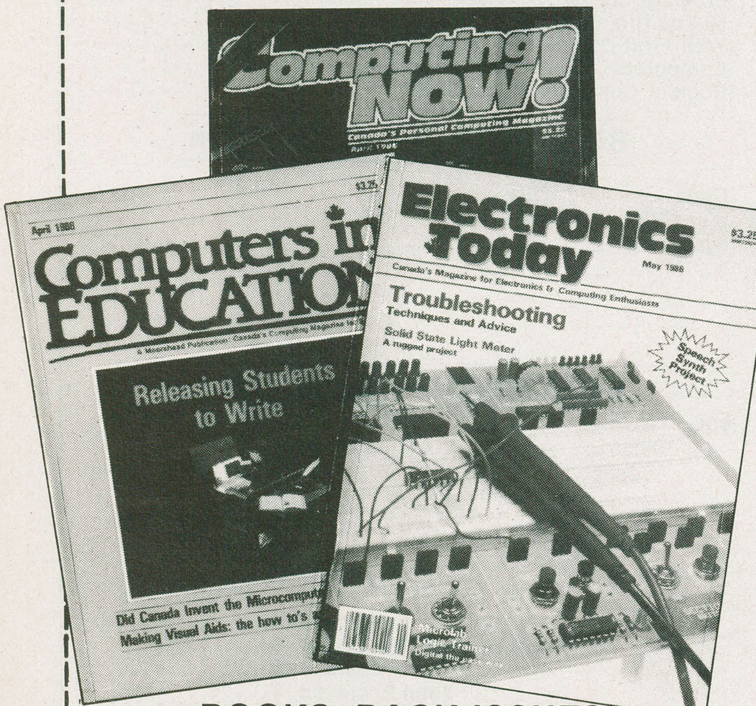
2D CAD ON A PC. IMSI Designer/Prodesign II, only 300! IN-A-VISION $649. Send cheque or money order to: **TECHNI-SOFT,** P.O. Box 261, Stn. P, Toronto, Ont. M5S 2S8 (416) 535-0863.

SOFTWARE, Hardware and Books at near 20% of Suggested Retail Price, available for most personal computers. Send for free catalogue. **COMPUTER DISTRIBUTORS,** P.O. Box 8217, Station F, Calgary, Alberta T2J 2V4. (403) 281-8445.

WHAT DO YOU DO?
Send us your typewritten or clearly printed words, your permanent address and telephone number and your payment (no cash please). Make your cheque or money order payable to Moorshead Publications. We're at 1300 Don Mills Road, Don Mills, Toronto, Ontario M3B 3M8.

WHAT DO WE DO?
We typeset your words (and put the first word and your company name in **BOLD** capital letters). Your advertisement will appear in the first available issue.

BE SURE OF YOUR ISSUE EACH MONTH. SUBSCRIBE TODAY.



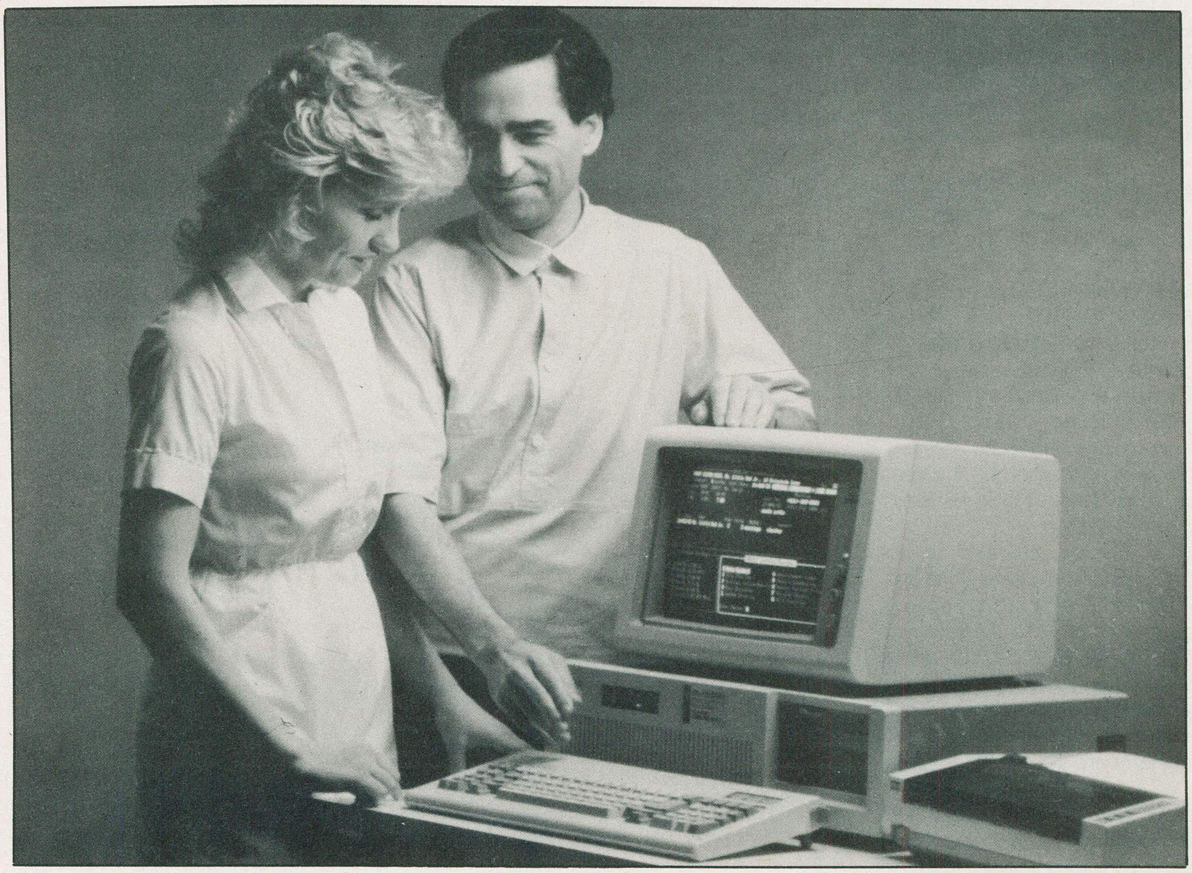
BOOKS, BACK ISSUES, SPECIAL PUBLICATIONS, BINDERS — SEE OVER

Moorshead Publications
1300 Don Mills Rd., Don Mills, Toronto, Ont. M3B 3M8

MERCHANDISE ORDER ☐ **Please fill out form overleaf**
SUBSCRIPTIONS: ☐ **NEW SUBSCRIPTION** ☐ **RENEWAL**
Electronics Today
☐ One year (12 issues) **$22.95** ☐ Two years (24 issues) **$37.95**
Computing Now! (Special Time-Limited Rate)
☐ One year (12 issues) **$14.95** ☐ Two years (24 issues) **$27.95**
Computers in Education
☐ One year (10 issues) **$25.00** ☐ Two years (20 issues) **$45.00**
Outside Canada (US Dollars)
For U.S. please add $3.00 per year ☐ other countries add $5 per year ☐

NAME ______
ADDRESS ______
TOWN/CITY ______ PROVINCE/STATE ______
CODE ______ DATE ______
POSTAL CODE ______
☐ Cheque enclosed DO NOT send cash
☐ Mastercard Account No. ______
☐ Visa Account No. ______
☐ American Express Account No. ______
Expiry Date ______
Signature ______

GrabIt for the IBM PC



Getting a copy of one's tube into a disk file, rather than a printout, isn't something that DOS is easily up for. It can be, though, with a bit of code.

by Steve Rimmer

The existence of a screen dump facility on the PC is one of its more enlightened low level routines. While quite a number of earlier computers had ways of getting this together, it was usually only by the most funky of means.

It would be thoroughly slick if DOS implemented a way of copying the screen to a disk file in a similar way, in much the same way as one can under the Macintosh's finder. Sadly, this isn't in the works. Pasting a screen dump of a spreadsheet into, say, a WordStar file, is rather beyond the whole effort.

The GrabIt program in this article is yet another resident interrupt handler, this time a solution to this little oversight. Run it, hit the shifted PrtSc key and a text file with the contents of the screen in it will turn up on your disk. It sounds blessedly simple and, in fact, it is. The code's pretty short.

Up For Grabs

Unlike as with terminal driven systems, such as many of the older eight bit computers, the screen data of the PC lives right on the memory bus. It's even stored in a reasonably useful form.

Depending on whether you have a monochrome or colour card, the screen data will turn up at segment 0B000H or 0B800H respectively. The first byte in the screen segment will represent the character in the upper left hand corner of the screen. You can prove this if you want to. Assuming you have a colour card, get into BASIC and run this little program

```
10 DEF SEG = &HB800
20 POKE 0,ASC("A")
```

The letter A should appear at the top of the screen.

If you now POKE 1,&H70, the A will turn up as a reversed out black character in a white block. The next byte over from the character is the attribute byte, determining the colour of the character byte immediately lower than it.

The fact of the character bytes being interleaved with attribute bytes is the first thing that makes copying the screen to a disk file a bit complicated. We have to sort out the characters, as the attribute bytes would make the text look decidedly peculiar.

The next tricky bit is that all the lines on the screen are of the same length. This seems fairly obvious, when one considers it, but it poses a problem in that all the lines of an ASCII text file rarely are. They end with the last printable character. A screen dump's lines would be padded out with spaces to make it eighty characters wide. This would make text copied from the screen a bit difficult to work with.

Finally, screen lines don't have carriage returns or line feeds at their ends. A screen dump would be a large block of really mangled text without them.

The way to get around most of this rather negative karma is to create a line buffer in the program that sorts out the screen. We can copy each line into it, and then manipulate it to find, for example, the actual end of the line by counting backwards from its end until a non-space character is found. The DOS two calls make it relatively easy to write random numbers of characters to a file.

The final problem is in the file that gets written to. Obviously, one can't really have a screen capture program that disrupts the screen by asking for a file name. In this case, the file that gets written to the first time the program is called will be SCREEN.A. The next time the interrupt comes down it will write SCREEN.B and so on. You might want to change this to something more sophisticated after you get the code working.

Into The Circus

The program itself is a pretty straight up resident interrupt handler. It traps the print screen interrupt, that's INT 05H, disabling the dumping of the screen to a printer and turning it instead into a screen capture command. Once the program has been run once, hitting the shifted PrtSc key will save the screen. You should see a momentary flicker of snow as the screen is copied, and a new file will turn up in whatever directory you're in at the moment.

The process of saving the screen data is pretty uninvolved. Everything happens through the line buffer. The STASH_LINE routine copies all the odd numbered bytes from the screen line pointed to by AX into the buffer. This encompasses a hundred and sixty bytes of screen RAM, resulting in eighty bytes of actual ASCII.

Screen Grab Utility

```
                COMMENT *
        Grab It - Screen grab utility

        copyright (c) 1986 steve rimmer

        "I used to be out of work and useless. Now,
        thanks to computers, I'm no longer out of work."
                                A.J. Sootlikker
                                Argasm Corp. Ltd.
                        *

VERSION         EQU     1        ;VERSION NUMBER
SUBVERSION      EQU     0        ;SUBVERSION NUMBER (BLOODY SUBVERSIVES)

SCREEN_WIDE     EQU     80
SCREEN_DEEP     EQU     25

STACK_SIZE      EQU     128      ;HOW MUCH STACK OVERHEAD
TRAP_NUMBER     EQU     05H      ;INTERUPT TO TRAP
TUBE_SEG        EQU     0B800H   ;SCREEN SEGMENT


                CODEX   SEGMENT
                ASSUME  CS:CODEX, DS:CODEX, ES:CODEX
MAIN            PROC    FAR
                ORG     100H

START:          JMP     INSTALL_HOOKS           ;JUMP OVER INTERRUPT HANDLERS

PAD_HANDLER:    PUSH    DS
                PUSH    AX

                MOV     AX,CS
                MOV     DS,AX                   ;SET UP LOCAL DATA SEGMENT

                MOV     [STACK_POINTER],SP
                MOV     [STACK_SEGMENT],SS      ;SAVE OLD STACK

                CLI
                MOV     AX,CS
                MOV     SP,OFFSET STACK
                MOV     SS,AX                   ;CREATE NEW STACK
                STI

                PUSH    ES                      ;SAVE CONTEXT
                PUSH    DX
                PUSH    CX
                PUSH    BX
                PUSH    AX
                PUSH    DI
                PUSH    SI
                PUSH    BP

                CALL    HANDLE_BYTE             ;DEAL WITH INTERUPT REQUEST
                POP     BP
                POP     SI
                POP     DI
                POP     AX
                POP     BX
                POP     CX
                POP     DX
                POP     ES                      ;RESTORE CONTEXT

                CLI
                MOV     SS,[STACK_SEGMENT]
                MOV     SP,[STACK_POINTER]      ;RESTORE OLD STACK
                STI

                POP     AX
                POP     DS

                IRET

STACK_POINTER:  DW      0
STACK_SEGMENT:  DW      0                       ;OLD STACK POINTER
LINE_INDEX:     DW      SCREEN_WIDE             ;LENGTH OF CURRENT LINE
LINE_BUFF:      DB      SCREEN_WIDE DUP (0)     ;BUFFER FOR CURRENT LINE
FILE_NAME:      DB      'SCREEN.'               ;FILE NAME
FILE_NAME1:     DB      'A',0                   ;FILE EXTENSION
HANDLE:         DW      0                       ;FILE HANDLE
NEW_LINE:       DB      13,10                   ;END OF LINE MARKER
                DB      STACK_SIZE DUP(0)       ;LOCAL STACK BUFFER
STACK:          DW      0

MAIN            ENDP

STUFF           PROC    NEAR

HANDLE_BYTE:    CALL    OPEN_FILE               ;OPEN THE FILE
                MOV     CX,SCREEN_DEEP          ;GET NUMBER OF LINES
                MOV     AX,TUBE_SEG
                MOV     ES,AX                   ;POINT TO SCREEN SEGMENT
HANDLE1:        PUSH    CX                      ;SAVE LINE COUNT
                MOV     AX,SCREEN_DEEP          ;GET LINE COUNT
                SUB     AX,CX                   ;
                MOV     BL,SCREEN_WIDE * 2      ;...TIMES LINE LENGTH
                MUL     BL                      ;POINT TO START OF LINE

                CALL    STASH_LINE              ;SAVE THE LINE TO DISK
                CALL    CHECK_LINE              ;SEE HOW LONG IT IS
                CALL    WRITE_FILE              ;WRITE THE LINE TO THE FILE
                POP     CX                      ;GET COUNT BACK
                LOOP    HANDLE1                 ;AND GO AGAIN
                CALL    CLOSE_FILE              ;CLOSE THE FILE
                INC     BYTE PTR [FILE_NAME1]   ;BUMP UP FILE EXTENSION
                RET

;THIS ROUTINE CHECKS TO SEE HOW LONG THE LINE IS
```

ANNOUNCING

Computers in EDUCATION 86

Hands-on Show and Conference

Arts Crafts Hobbies Building
Exhibition Place, Toronto
October 23rd, 24th, and 25th, 1986
Thursday and Friday 10:00 - 9:00
Saturday 10:00 - 5:00

- A professional show for Canadian educators emphasising personal hands-on evaluation of educational software.
- A huge number of programs covering the entire educational spectrum will be available for private examination in the Computer Lab Facility.
- Computer Lab Facility will comprise over 100 educational microcomputers for unpressured personal testing of either the software or hardware.
- Demonstrations of the latest educational hardware.
- Wide range of classroom-orientated peripherals on show.
- Full programme of educational computing seminars.

For Exhibitor Information contact:
Arthur Nagels
1300 Don Mills Road, Don Mills, Ontario, M3B 3M8
(416) 445-5600

Circle No. 31 on Reader Service Card.

The end of this line exists at the point where all further characters in the line are spaces. The easiest way to find this is to point to the end of the buffer and keep working back until we're either pointing to the start of the buffer... suggesting that we have what was a blank line on the screen... or until we find a non-space character. The CHECK_LINE gets this together, setting the value of LINE_INDEX to the actual length of the line.

The WRITE_FILE routine sends the line out to the disk file. It also sends a carriage return and a line feed at the end of each line.

Finally the program closes the file and increments the byte that makes up the file extension, so that A becomes B, and the next time the program is called it will write to a different file. You can have quite a number of screen files, renaming them later if you want to.

One potential hassle in the file naming system I've used here is that the files will always turn up in the subdirectory that you're currently in. You can have them all written to a designated subdirectory. For example, to have them turn up in the NEWTFARM subdirectory... you do have a NEWTFARM subdirectory, of course... change the string at FILE_NAME to "\NEWTFARM\SCREEN.".

This is a pretty simple bit of code... if one were to expand on it, it should probably have it open a window on the screen when it's hit to ask for a file name, and incorporate some mechanism to disable itself for when you want to use the real screen dump mechanism. However, these things take an inordinately large amount of code to get together... I think that programs with simple functions should be reasonably painless to type in.

No Name Apocalypses

There are a number of fairly practical uses for this thing. You can have it create screen images of spread sheets or debug sessions to later include in documents... reports or manuals. You can use it to capture part of a debug session to ultimately use the hex data somewhere else... as I did in the RAMSET loader for the PC memory article elsewhere in this magazine. You can also just use it to get screen dumps if your printer happens to be interfaced to your PC and, as such, won't behave with the BIOS's print screen routine. Boot this little mud shark and you'll have printable files that can be sent to a non-standard printer with grace. **CN!**

```
CHECK_LINE:     MOV     BX,OFFSET LINE_BUFF + SCREEN_WIDE
                MOV     [LINE_INDEX],SCREEN_WIDE
CHECK_0:        DEC     BX
                MOV     AL,[BX]                 ;CHECK FOR SPACE
                CMP     AL,' '                  ;IF IT AIN'T
                JNE     CHECK_1                 ;WE HAVE END OF LINE
                DEC     [LINE_INDEX]            ;ELSE BUMP DOWN LINE LENGTH
                CMP     [LINE_INDEX],0          ;SEE IF IT'S AN EMPTY LINE
                JG      CHECK_0                 ;LOOP 'TIL DONE
CHECK_1:        RET

;THIS ROUTINE SAVES THE LINE POINTED TO BY AX INTO LINE BUFFER
STASH_LINE:     MOV     CX,SCREEN_WIDE          ;GET WIDTH OF SCREEN
                MOV     SI,AX                   ;GET POINTER INTO SCREEN
                MOV     DI,OFFSET LINE_BUFF     ;POINT INTO LINE BUFFER
STASH1:         MOV     AL,BYTE PTR ES:[SI]     ;GET A BYTE
                MOV     BYTE PTR [DI],AL        ;SAVE IT IN BUFFER
                INC     SI                      ;SKIP TO NEXT BYTE
                INC     SI                      ;...OVER ATTRIBUTE
                INC     DI                      ;POINT TO NEXT BYTE IN BUFFER
                LOOP    STASH1                  ;AND LOOP
                RET

;THIS ROUTINE WRITES THE CONTENTS FOR THE LINE BUFFER
WRITE_FILE:     MOV     CX,[LINE_INDEX]         ;GET THE LINE LENGTH
                MOV     AH,40H
                MOV     DX,OFFSET LINE_BUFF     ;POINT TO BUFFER
                MOV     BX,[HANDLE]             ;GET HANDLE
                INT     21H                     ;WRITE IT
                MOV     CX,2
                MOV     AH,40H
                MOV     DX,OFFSET NEW_LINE
                MOV     BX,[HANDLE]
                INT     21H                     ;AND WRITE CRLF
                RET

;THIS ROUTINE OPENS THE FILE
OPEN_FILE:      MOV     DX,OFFSET FILE_NAME
                MOV     CX,0
                MOV     AH,3CH
                INT     21H
                MOV     [HANDLE],AX
                RET

;THIS ROTUINE CLOSES THE FILE
CLOSE_FILE:     MOV     AH,3EH
                MOV     BX,[HANDLE]
                INT     21H
                RET

END_HANDLERS:   DB      0

;THIS CODE HOOKS IN THE INTERUPT VECTORS
INSTALL_HOOKS:  CLI                             ;DISALLOW INTERUPTS
                PUSH    DS
                MOV     AH,25H
                MOV     AL,TRAP_NUMBER
                PUSH    CS
                POP     DS
                MOV     DX,OFFSET PAD_HANDLER
                INT     21H                     ;INSTALL HOOK INTO
                POP     DS                      ;LOW MEMORY VECTOR TABLE
                STI

                MOV     AH,9
                MOV     DX,OFFSET MESSAGE
                INT     21H                     ;SAY "HOWZIT GOIN'"

                MOV     DX,OFFSET END_HANDLERS  ;POINT TO OUR PROTECTED CODE
                INT     27H                     ;TERMINATE BUT STAY RESIDENT

MESSAGE:        DB      'GrabIt - Screen Text Grabber Program '
                DB      VERSION+'0','.',SUBVERSION+'0'
                DB      13,10,'Copyright (c) 1986 Steve Rimmer$'

STUFF           ENDP
CODEX           ENDS

                END     START
```

BUSINESS DIRECTORY

Business · Business · Business · Business · Business · Business

ELECTRONIC SHOPPING

FREE SHOPPING BY MODEM
SATELLITE ELECTRONIC SHOPPING MALL

Take advantage of this first FREE electronic shopping mall. Available to all computer and modem users. Now you can shop for insurance cars, computers, books, software, travel tickets & holidays, and much more in the comfort of your own home or office. The electronic stores are open 24 hours a day, 7 days a week for your convenience.

STORES IN THE MALL	MODEM
Bayside Computerized Insurance Shopping Data Base	831-0666
Compu-Car (Shop for new GM, Chrysler and Ford cars)	831-6678
Computer Shoppe (Order computer hardware)	831-6907
Alpha Books & Software (Order computer books and software)	831-6908
Electronic Travel Agency (Book flights, holidays & hotels)	Coming soon

And More Openings Coming

Check with any of the shops for more mall information.

LINK UP WITH YOUR SATELLITE DEALERS TODAY!!

Circle No. 12 on Reader Service Card

ENGINEERING SOFTWARE

HANDIDAT

For Engineers and Scientists

- Unit conversion – physical constants
- Thermal couple tables (T, J, K, E)
- Vapour pressure of ice and water
- Enthalpy of liquid water, saturated steam and super heated steam

POP-UP PROGRAM FOR IBM PC'S AND COMPATIBLES

Send cheque or money order ($1.75) to:

J.B. LOGI-SOFT 10508 75th Ave., Edmonton, Alta. T6E 1J4

Circle No. 13 on Reader Service Card

IBM PC SOFTWARE

ANDSOR

The Andsor Collection™

The Andsor Collection is a unique concept: it creates a complete, self-contained, window-based data management environment, in one DOS file. This simplifies everything. You can easily combine functions to create your own solutions in any application: calculations, database management, modeling, text processing, charts, data analysis, statistics, reports, labels, forms, presentations, mail-merge, etc.

The Andsor Collection helps you make the most of any configuration, from a 128K PC*jr* to a PC AT. It comes with a superb, 400 page hard-cover manual, with many examples. From simple calculations, files, inquiries, to complex models, data structures, reports: when your favourite data manager / spreadsheet / word processor / integrated system cannot provide the solution you need, remember *The Andsor Collection*.

$135 + $5 s&h *(in Ontario add 7% sales tax)*

60-day money-back guarantee

Hardware requirements: IBM PC/XT/AT/PC*jr* or fully compatible, 128K minimum, one drive or hard disk, monochrome and/or colour monitor, DOS 2.0+. *Not copy-protected.*

Visa / MasterCard / AmEx / Cheque / Money Order / COD

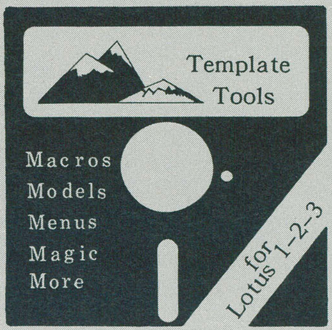
To order, or for more information, call or write now:

ANDSOR RESEARCH INC.
181 University Avenue, Suite 1202
Toronto, Ontario M5H 3M7

(416) 364-8423

Circle No. 14 on Reader Service Card

BUSINESS SOFTWARE



A $95.00 MIRACLE

MOUNTAIN BUSINESS SOFTWARE

1-403-453-5972

VISA MasterCard

16010 B
– 118 Avenue
Edmonton, AB T5V 1C6

Circle No. 15 on Reader Service Card

SOFTWARE-C PRODUCTS

C Products from Lifeboat

Phoenix Tools		Aids & Tools		Lattice	
PRE C	$475	Greenleaf	$260	C. Compiler 3.0	$600
P-Fix Plus	$475	Smorgasbord	$180	Windows	$355
P-Finish	$475	Panel	$355	Basic Library	$245
P-Mate	$270	MSD Debug	$235		
Interpreters		**Graphics**		**File Access**	
Run/C	$180	Dr. Halo	$125	B-Trieve	$345
Instant-C	$675	Multi-Halo	$350	C-Tree	$485
				DBC	$325
				Misc	
				8087-3	$190
				Float-87	$250
				Desmet C	$185

*etc *etc *etc

SCANTEL SYSTEMS LTD.
(416) 449-9252

(Dealer enquires invited)
801 York Mills Rd., Don Mills, Ont.

Circle No. 16 on Reader Service Card

MONEY-MAKER

THIS SPACE IS FOR SALE

1 time insertion	$199.00 CND.
3 time insertion @	$189.00 CND.
6 time insertion @	$179.00 CND.
12 time insertion @	$169.00 CND.

For more information or to place an AD
Call WAYNE FLEET (416) 445-5600

COMPUTER & DESK DRIVE REPAIR

COMPUTER SERVICE
ONSITE OR CARRY-IN

We service most major brands including:

IBM XT/AT and compatibles	TEXAS Instruments
Burroughs	Convergent Technologies
C.ITOH	Contel Cado
N.C.R.	Mannesmann Tally

We also service most Floppy and Hard Disk Drives

For others call for quotation:

(416) 827-5040

Network Data Systems (S.W.O.) Ltd.
2190 Speers Rd., Oakville, Ontario L6L 2X8
Servicing Southwestern Ontario for over 8 years

Circle No. 17 on Reader Service Card

Seven Ways You Can Add Equity to Your Business.

EQUITY I
Configuration 1
The first configuration, ideal as an entry level system, second office computer, or for home use, consists of a single 360KB floppy disk drive and 256KB of RAM.

EQUITY I
Configuration 2
If you need more muscle, step up to the dual floppy disk drive, 256KB RAM system. With its 720KB of disk storage, this configuration is ideal for more intensive word processing, spreadsheets and telecommunications activities.

EQUITY I
Configuration 3
For a real boost in performance, you'll love the single floppy drive with the addition of a 20MB internal hard disk. It's a combination with the power to tackle business applications such as accounting and database management.

With the remarkable family of Equity personal computers, Epson can make your selection easy.

EQUITY II
Configuration 1
The first configuration boasts a single 5 1/4-in., 360K floppy disk drive and 640K RAM memory. It packs enough punch to give you the edge across a broad range of business activities.

EQUITY II
Configuration 2
For even more power, move up to the second configuration. Its single 360K floppy disk drive with 20MB internal hard disk is ideal for everything from heavy duty number crunching to storing reams of information.

EQUITY III
Configuration 1
Choose the 1.2MB 5 1/4-in . drive with 640K of memory and put higher applications to work. The Equity III runs off a powerhouse 80286 microprocessor at 6 MHz that can recalculate large spreadsheets and retreive files in a flash. Its 640K standard memory is expandable to 15.5MB, to take full advantage of the eight full-sized IBM-compatable option slots.

EQUITY III
Configuration 2
If you really need to push productivity beyond the capabilities of any ordinary computer, select the outstanding 20MB internal hard disk configuration. With it, you can customize your system to store thousands of pages of information. You may even choose a 40MB internal hard disk.

Epson Canada offers a Canadian Solution. We have sales and service locations across Canada. All Epson products have a standard 1-year warranty.

EPSON®
EPSON CANADA LIMITED
Circle No. 19 on Reader Service Card

For Your Authorized EPSON Dealer Call:

VANCOUVER (604)731-4166	EDMONTON (403)428-0318	WINNIPEG (204)895-2692	OTTAWA (613)726-9335	QUEBEC CITY (418)654-4707
CALGARY (403)255-2772	SASKATOON (306)665-3399	TORONTO (416)495-1049	MONTREAL (514)331-7534	HALIFAX (902)455-0817

EPSON PRODUCT CENTRES
SCARBOROUGH (416)296-9898
TORONTO (416)364-1143
MISSISSAUGA (416)896-0500

Epson Product Centres are owned and operated by Epson Canada Ltd.

TRANSPAC

FOR THE 4 IN 1 SUPERCOMPUTER

THAT IS MADE IN CANADA AND COMES WITH A ONE-YEAR WARRANTY FROM XEROX® SERVICE CENTERS THROUGHOUT CANADA

Monitor not included

THE SUPER FAST TRANSPAC II

CSA approved 150 watts power supply, genuine Keytronics Keyboard, 256K RAM, 2 serial and one parallel ports, real-time battery back-up clock plus speed selectable via external switch. 4.77 and 8MHz uses 8088-2 processor.
(Most software will run on the higher speed.)

$1725.00

Other configurations

- TRANSPAC III
  10 Meg Hard Drive/1 Floppy/256K **$2486.35**
- TRANSPAC IV
  10 Meg Hard Drive/2 Floppy/256K **$2649.00**
- TRANSPAC V
  20 Meg Hard Drive/1 Floppy/256K **$2704.88**
- TRANSPAC VI
  20 Meg Hard Drive/2 Floppy/256K **$2867.53**

SUPERB IBM PC & XT COMPATIBLITY AND MORE...

Not just Hardware but also Software compatibility. Supports PC-DOS and CPM/86. Will run all your favourite software such as Lotus 123, Autocad, Flight Simulator, Dbase III etc.

MULTIFUNCTION FEATURES COMES AS STANDARD

- 2 Serial Ports (not just 1)
- 1 Parallel Port
- Real Time, Battery Back-up, Clock
- Memory expansion on-board to 1 megabyte
- 64K and 256K chips can be mixed
- Optional 8087-2co-processor
- Sasi Hard Disk Interface
- 54K User Definable Rom expandability
- Low Cost Modem Interface
- Ram Disk software included

FLOPPY CONTROLLER CARD COMES AS STANDARD

Will Support Up To 4 Floppies

HIGH-SPEED BOARD COMES AS STANDARD

Speed Selectable Between 4.77 MHz and 8 MHz via external switch.

Note: Up to now all these features were only available by purchasing expensive interface cards. Now with the ACS-1000 single board supercomputer (see reviews in March and May Issues of "Electronics Today") all these features come as standard on one board. What's more, since the ACS-1000 is a product of Top-Down integration all these functions will work together as never before. And, more importantly, since everything is already on the main board the remaining expansion slots will give you true expansion capability.

PLUS MORE

- genuine Keytronics Keyboard
- reset switch
- speed selection via external switch
- 150 watt CSA approved power supply
- quiet slim-line DS DD 5¼ 360K disk drives
- colour video adaptor included. RGB and composite board or Hi-Res monochrome board. (customer choice)
- ONE YEAR WARRANTY
- even after all the above standard features you still have 6 expansion slots
- made in Canada
- 256K RAM standard minimum

WARRANTY

The **TRANSPAC** computer not only offers you high quality and high performance at low cost but also gives you one full year's service warranty — parts and labour — right across Canada from Xerox, one of the most reputable service organizations in Canada. This is possible because of the extra care we take in sourcing our components and our quality control. For example, every computer that leaves our site undergoes in total 72 hours of burning-in. This ensures a very high reliability rate in the field and of course, satisfied customers.

TRANSPAC
TRADING CORP.
for the computer that is not just another look-a-like

5485 — 128th St., Surrey, B.C. Canada V3W 4B5
Order line only 1-800-663-0073 Telex: 04-365628
Master Charge & Visa, Certified Cheque, COD, Money Order accepted.
Shipping & Insurance extra.

Circle No. 20 on Reader Service Card